

微软公司 华人专家鼎力之作

软件开发的科学与艺术

The Science and Art of Software Development

陈宏刚 林 斌 凌小宁 张益肇 熊明华 张亚勤 著

- 全面总结软件开发思想
- 深入解析软件开发过程



电子工业出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY

<http://www.phei.com.cn>

软件开发的科学与艺术

The Science and Art of Software Development

在现代软件开发热潮中,微软在二十几年的发展过程中形成了其独特的软件开发与设计的企业文化。软件开发是一门科学,更是一门艺术,尤其是对于微软这个软件王国。

——微软亚洲研究院院长兼首席科学家

陈宏

本书是国内第一本由微软公司华人专家编写的介绍软件研发经验的中文书,希望通过本书的出版对中国的软件教育和软件企业的发展起到很好的促进作用。

学习成功者的经验会成为您成功的捷径;

吸收开拓者的教训会有助您正确的抉择。

——电子工业出版社社长

王志刚

这是一本不可多得的正面剖析微软软件开发过程的宝贵资料,可供广大从业者借鉴学习,提高自身开发水平。

——著名软件开发专家、豪杰公司总经理

梁肇新

国内介绍软件开发与管理,并具有实际指导意义的书籍是很罕见的,而这本书组织了多名微软的华人专家,以其亲身经历,从多个角度向读者揭示了微软公司软件开发与管理的秘密。非常精彩,值得中国软件界人士仔细阅读。

——WWW.CSDN.NET 创始人、《程序员》杂志社社长

茹一昂

ISBN 7-5053-7555-5



9 787505 375550 >



责任编辑:郭立

施玉新

版式设计:朱仁平

封面设计:张昱

本书贴有激光防伪标志,凡没有防伪标志者,属盗版图书。

ISBN 7-5053-7555-5/TP·4380 定价:38.00元

TP311.52

26

微软公司华人专家鼎力之作

软件开发的科学与艺术

The Science and Art of Software Development

陈宏刚 林 斌 凌小宁 张益肇 熊明华 张亚勤 著

北方工业大学图书馆



00512317

电子工业出版社

Publishing House of Electronics Industry

北京·BEIJING

内 容 简 介

本书凝聚了微软公司多位专家在多年研究和工作中获得的宝贵经验,并通过对许多成功或失败案例的中肯剖析,为读者展现出软件开发的思想与过程。作者认为,软件开发既是一门科学,需要尽可能地量化;又是一门艺术,需要有经验的人来把握。本书还以华人专家的视角透视了微软公司的企业文化。这些内容将启迪读者的软件开发思想观念,对中国现代软件产业的发展和进步也具有重要的借鉴意义。

本书是软件从业人员案头不可或缺的珍藏书,也可作为高等院校师生的教学参考书。

未经许可,不得以任何方式复制或抄袭本书之部分或全部内容。
版权所有,侵权必究。

图书在版编目(CIP)数据

软件开发的科学与艺术/陈宏刚等著. —北京:电子工业出版社, 2002.5

ISBN 7-5053-7555-5

I. 软… II. 陈… III. 软件开发 IV. TP311.52

中国版本图书馆 CIP 数据核字 (2002) 第 019617 号

责任编辑:郭立 施玉新

印刷:北京天竺颖华印刷厂

出版发行:电子工业出版社 <http://www.phei.com.cn>

北京市海淀区万寿路 173 信箱 邮编 100036

经销:各地新华书店

开本:787×980 1/16 印张:23.25 字数:346 千字

版次:2002 年 5 月第 1 版 2002 年 5 月第 1 次印刷

印数:15 000 册 定价:38.00 元

凡购买电子工业出版社的图书,如有缺损问题,请向购买书店调换。若书店售缺,请与本社发行部联系。联系电话:(010) 68279077

微软公司副总裁

李开复

从1998年回到中国创办微软中国研究院，到2000年调回微软总部出任负责自然交互式软件部门的副总裁，我接触了大量的中国软件企业。他们经常问到的一个问题是：微软是如何保证像Windows这样大型的软件开发项目能按照预定的时间发布，并能同时保证产品质量的？我带着这个问题回到总部的产品部门，通过实际地指导微软的软件产品开发，我认识到，中国的软件企业所寻找的是的一种能够有效地指导软件开发的模式和经验。

我们可以很自然地将软件分成两类：软件产品和软件项目。软件产品指的是不局限于特定领域的、可以被广大用户直接使用的软件系统，如微软的Windows、Office等等。这类系统的特点是技术含量高，开发时要考虑到各种不同的用户需求。软件项目指的是针对特定领域提供优化业务流程的软件系统，如我们常说的管理信息系统(MIS)和电子商务系统。这类软件的特点是领域知识所占的比重较大，相对技术性而言，工程性更强。

针对这两种不同类型的软件，当然应该有不同的软件开发方法去指导项目开发过程。针对软件项目的开发，目前比较成熟的软件开发方法是由卡莱基梅隆大学软件工程学院提出的软件成熟度模型

1/5/01/08

(CMM)。这种软件开发模型试图将整个软件开发过程规范化和量化，直到可以对软件开发过程进行定量的控制和优化。

微软的软件开发经验更多的是集中在软件产品的开发上。软件产品的开发随着技术的发展和用户需求的变化，面临着更多的变数和挑战。在软件产品开发过程中，理想中的量化和优化其实是不太现实的。软件产品的开发更多地依赖于优秀的和富有效率的开发团队、具有长远眼光的项目定位、灵活而有效的项目管理、保证产品质量的软件测试和处理各种风险和突发事件的机制。

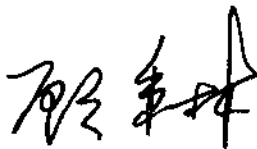
如此看来，软件产品的开发既是一门科学，需要尽可能地量化；又是一门艺术，需要有经验的人来把握。如何在两者之间取得动态的平衡是成功研发软件产品的关键。微软公司在27年的软件产品开发的实践中积累了大量的经验，这是微软最大的财富，无疑也是中国软件企业很好的参考。

《软件开发的科学与艺术》一书最有价值的一点是作者都是在微软从事多年软件研发的中国员工，他们在比较了中国软件企业和微软公司的软件开发过程的基础上，结合自己的工作体验和思考，对如何把握软件开发的科学性和艺术性做出了经验性的总结。这些经验对中国软件企业来讲，不一定能够直接应用；但它们更像一把钥匙，能激发中国的软件企业通过自己的实践，找到适合自己的软件开发方法。

我衷心地希望中国的软件产业能够迅速进入健康的发展轨道，与写此书的在微软工作的中国员工一样，我希望中国能早日出现自己的明星软件企业。

2002年3月

中国科学院院士
清华大学常务副校长
软件学院院长



软件产业是进行软件产品开发、生产、销售和信息技术服务的技术产业，是信息产业的灵魂。发展软件产业已成为我国国民经济发展战略的一个重要组成部分。

世界经济发达国家硬件与软件市场的比例基本持平，而目前中国软件市场还不及硬件市场的20%。软件对信息化推进的严重制约已经引起我国政府的高度重视，当然这对于我国的软件企业而言也意味着巨大的发展机会。那么，我国发展软件产业的难点在哪里呢？

软件以智力和知识为基础的生产方式决定了它较之其他产业更强调人才的重要性。软件产业的竞争越来越集中到对人才的竞争上。应该说，我国软件开发方面的人才并不少，特别是在校的大学生，软件编程能力很强，但这种优势并没有在我国软件产业的发展中显现出来。这一方面是由于我国激励机制不完善以及待遇相对较低，使得软件人才纷纷外流；另一方面，国内对软件人才的培训和教育方式也滞后于软件产业发展的需要。具体地说，就是教育与产业和市场需求脱节。我们每年有大量的毕业生，但因为所学不能适合软件产业岗位的需要而不得不在进入企业后再培训，这大大延缓了整个

软件产业发展的速度。现代软件产业的发展要求具有专业知识的软件工程师，熟悉软件工程规范的系统分析员，具备现代企业管理经验的软件企业家，而这些都是目前我国软件人才培养和教育中所欠缺的。这也是我国政府在全国各大学范围内建立软件学院的初衷。微软亚洲研究院此时推出这本书对于当代软件人才的自身培养和提高都是大有裨益的。

《软件开发的科学与艺术》一书基本涵盖了现代软件开发的全过程，内容涉及软件产品的概念设计、进程管理、代码编写、程序测试和文档整理等各个方面。特别值得一提的是，他们都是微软公司资深管理人员与工程师在自身实践的过程中的亲身体会和总结，其中大部分案例都是他们在微软公司软件开发过程中的真实环境下遇到的问题和解决方法。关于软件测试等软件工程的新兴领域的阐述和研究更值得我国软件企业学习。我想，本书对国内软件企业的发展有一定的帮助和借鉴作用。

目前，我国正在大力推行软件产业，随着政府对投资环境的改善，对知识产权保护力度的加大，软件产品市场的进一步规范，加上我国软件人才的勤奋努力，相信中国的软件产业必将迎来一个崭新的明天。

2002年3月

中国软件行业协会理事长

杨天行

什么将影响我们的未来生活？什么将决定国家的未来竞争力？毫无疑问，答案之一就是信息技术。

科学技术的突飞猛进，知识经济已初见端倪。信息技术在知识经济的发展过程中处于中心位置。为了未来的繁荣和发展，我们必须关注信息技术。当今，信息技术正推动社会日新月异地发展，信息技术已经影响到我们的生活方式、思维观念等。

作为信息技术的核心，软件开发正日益受到人们的重视。软件开发者是软件开发的基础，他们的素质将决定软件的开发水平。究竟我们需要什么样的开发者？什么样的开发者才能肩负弘扬民族软件的重任？这是我们的教育必须面对的一个严肃的课题。我们必须正确培养出大批掌握信息核心技术的开发者才能迎接未来之挑战。

掌握正确的学习思路和方法为什么重要，有了正确的学习思路和方法，才能更好地分析各种问题，寻找解决问题之道。

向成功人士学习是迈出成功的第一步。正是基于此，我阅读了电子工业出版社的《软件开发的科学与艺术》一书，甚是欣赏其内容。本书汇集了微软专家所做的“软件开发的科学和艺术”的讲座精华，使我们能站在成功者的肩膀上，分享他们的开发经验、感悟，

高屋建瓴地去认识问题，启发思维，少走弯路。

我相信，本书的出版会对广大软件开发者有所帮助和指导。也让我们一起为中国的软件早日走向世界而努力！

2002年3月15日



陈宏刚

微软亚洲研究院商务及高校关系高级经理,主管微软亚洲高校关系、微软亚洲研究院人力资源和商务计划及发展等。在微软总部工作期间,参加过 Windows 95, Exchange Server 4.0和4.5, Internet Explorer 4.0和5.0, SQL Server 2000 的开发和测试。



林 斌

微软亚洲研究院新技术开发部经理。通过长期的工程实践,摸索了一整套编写程序代码体系。在微软总部期间,先后在不同部门担任软件开发高级工程师(SDE),在互联网商业服务组和交换协议组担任软件开发组长。



凌小宁

曾在微软公司多个产品和研究部门担任软件设计工程师、软件开发组组长、软件开发部经理、软件开发总工程师和项目管理经理。在图形学、多媒体及用户界面方面获多项美国技术专利,并发表过多篇计算机人工智能和数据库方面的学术文章。



张益肇

微软亚洲研究院主任研究员。主要研究方向是自然语言理解、机器学习和信号处理。目前在研究院主要从事自然语言理解方面的研究工作。



熊明华

微软公司项目经理。参与过Java VM, Internet Explorer, Windows Me, Windows 2000, .NET My Service等产品的开发管理工作。目前在MSN部门从事项目技术总体设计工作。



张亚勤

微软亚洲研究院院长兼首席科学家。数字影像和视频技术、多媒体通信及Internet方面的世界级专家。美国电气与电子工程协会院士(Fellow of IEEE), 1998年获得美国电子工程师荣誉学会授予的“杰出青年电子工程师奖”。



导 读

《软件开发的科学与艺术》成书的原因与过程	1
《软件开发的科学与艺术》的内容	3
本书的意义	4

⇒ 1

第1章 全球软件产业现状、趋势与挑战

⇒ 5

The Challenges and Opportunities of the Global Software Industry

软件是一台计算设备的思维中枢。经过数十年的发展,软件产业已经成为当今世界投资回报最高的产业之一,而这一产业正在潜移默化地改变着我们赖以生存的这个星球的面貌。新世纪,软件产业已经呈现出了一些引人入胜的转变迹象,本章概括地把这些迹象表述为三个趋势:网络化、服务化与全球化,并从技术角度就高科技产业所面临的机遇与挑战阐述了作者的观点。

1.1 软件产业的网络化趋势	8
1.2 软件产业的服务化趋势	11
1.3 软件产业的全球化趋势	13
1.4 网络化、服务化、全球化趋势对中国软件产业的启示	15

第2章 现代软件开发对人才的要求

⇒ 19

Talent Beyond Technology—What Kind of Talent We Need for Modern Software Development

软件开发是一门科学,更是一门艺术。微软在二十几年的发展过程中形成了其独特的软件开发与设计的企业文化。目前,中国软件业的发展喜忧参半。中国拥有高素质、基础扎实、学习能力强且思维敏锐的软件专业人员,但中国大规模的软件生产尚处于初期发展阶段,软件的研究与开发过程中尚有许多亟待解决的问题。本章中,凌小宁博士关于微软企业文化以及软件开发人才等方面作了精辟的论述,这些内容是我们中国学生及软件从业人员应该了解、理解并恰到好处地遵循的。

2.1 现代软件开发概述	21
2.2 独具魅力的微软企业文化与软件开发人员的培养	23
2.3 从差别中寻找解决方案	28
2.4 扎实的基础和创新、独立的工作能力	33
2.5 主人翁精神和团队精神	34
2.6 锲而不舍、从错误中学习的精神	46

第3章 从研究到产品

► 51

From Research to Products

如何将研究成果投向市场并获得成功,这是许多人关心的问题。实际上,从研究成果到产品是一个非常复杂并且非常漫长的过程,其中会牵涉到相当多的问题。张益肇博士根据多年来积累的丰富实践经验,首先介绍了一种关于产品空间的思维方式——技术生命周期,以及在生命周期各阶段中用户对技术和市场的影响;其次通过具体的案例说明技术和市场的关系;最后总结出了三条宝贵的规则,相信会给读者带来很大的启发。

3.1	引言	53
3.2	技术生命周期	56
3.3	案例分析	62
3.4	练习	67
3.5	间断技术	73
3.6	基本规则	83
3.7	推荐书目	87

第4章 微软的软件开发

► 89

Software Development at Microsoft

现在的软件开发不再是个人英雄主义打天下的时代了,尤其是像微软这样大的软件公司,一个软件都是由几百人甚至几千人共同合作完成的。那么如何管理这样庞大的开发阵容?员工是如何分工的?他们之间又是如何协作的?这些都是大家关心的问题。陈宏刚博士结合自己在微软公司的亲身体验,并结合具体实例,从一个较高层次介绍了微软的产品团队、软件开发过程和开发方法。

4.1	概述	91
4.2	微软的产品团队	94
4.3	微软的软件开发过程	106
4.4	想法和意图批准里程碑	112
4.5	产品计划的通过里程碑	118
4.6	范围完成/第一次使用里程碑	123
4.7	发布阶段	130

Source of the Software Design

在有些人眼里,今天的软件开发似乎已成为简单的事件:已有了不少很好的开发工具和软件库,软件开发人员训练有素,都强烈渴望去编写很酷的软件,可以在几天的时间里编写出一个相当复杂的软件。但为什么有一些软件能够得到用户的喜欢,而另一些则不能?为什么有些软件能够在市场上成功,而有些则受到冷落?由此可见,开发软件并不一定难,难就难在如何开发有用的软件。本章,凌小宁博士就根据自己多年的实践经验,回答“如何设计有用的软件”这个问题。

5.1	软件设计简述	143
5.2	三个困难的问题	144
5.3	设计之源	145
5.4	错误设计之源	149
5.5	基于用户情景的设计	153

Program Management

项目管理是一种广泛应用于各种工程、金融甚至农业生产中的技术管理过程。在IT行业,项目管理常常是决定产品或企业能否成功的最重要指标之一。中国历经了15年的不懈努力,加入世界贸易组织终成现实,这为我们带来了前所未有的机遇和挑战。我国政府所属各部门和企业领导对于项目管理也越来越重视,现在市场上名目繁多的各类项目管理培训就可见一斑。

熊明华在微软担任项目经理这一职位多年。本章是根据他亲身实践,对微软的项目管理进行了详细的介绍,相信对我国软件业项目管理水平的提高会有所促进。

6.1	项目管理简述	163
6.2	什么是项目经理	164
6.3	项目经理的行政结构与工作关系	167
6.4	为什么需要项目经理	175
6.5	项目经理每天的具体工作是什么	178
6.6	做项目经理的背景要求	182
6.7	结论	187

第7章 写好代码的十个秘诀

► 189

10 Things You Can Do To Write Better Code

在现代软件的开发中,如何写出具有正确逻辑而且执行速度快的代码是众多的软件开发人员所追求的目标。林斌在微软总部担任了多年软件开发高级工程师,参加了微软多种产品的开发工作。本章,他根据自己多年的亲身体验,为读者总结出一流代码应该具备的特性,以及写一流代码的十个秘诀。文中展示了丰富的具体代码实例,并进行了详尽透彻的分析,最后提供了正确的解决之道。如果你是一名软件开发人员,相信本章对你今后的编程工作会有很大的帮助。

7.1	简介	191
7.2	编写代码的十大秘诀	196
7.3	结束语	228

第8章 如何提高程序的性能

► 231

Secrets of Software Performance

如果你是一名软件开发人员,是不是经常因为程序的性能而受到老板的指责?你是不是经常因为程序的性能而被用户刁难?你是不是经常因为程序的性能而饱受等待之苦?那么,怎样才能尽量优化应用程序,提高其性能呢?本章,林斌根据自己多年在微软进行软件开发的实践经验,为我们提供非常棒的提高性能的方法,并列举了具体的案例学习,最后,还详细介绍了令无数软件开发人员头痛的内存问题。如果你是一名软件开发人员,阅读本章后,立即将这些方法应用到你的应用程序中,体验一下性能提高的喜悦吧。

8.1	提高性能的方法	233
8.2	案例学习	234
8.3	内存	241

软件测试是一门非常崭新的学科,目前研究的内容还不很深入,所涉及的只是测试数字、测试函数等一些非常简单的问题,可以说还处于婴儿阶段。由于软件测试学科还不成熟,它到底需要一个什么样的专业基础,尚无定论,而且目前还没有一种很好的标准来衡量一名测试人员的优劣。本章,陈宏刚博士根据亲身体会,以微软公司为例讲述软件测试的方法及实施过程。陈宏刚博士指出,软件测试学的发展还有赖于大家共同努力,促进其研究的不断深入。相信本章能对软件测试工作起一定的指导作用

9.1	概述	257
9.2	关于 Bug	266
9.3	软件测试方法和辅助工具	269
9.4	相关测试文档	276
9.5	如何与项目经理及开发人员沟通	278
9.6	结束语	286

正因为软件测试还是一门新兴学科,各方面的规范还不完备,目前软件测试工作还无一定的规范可依据。微软经过二十余年的发展,在实践中不断探索和总结出一套独特的软件测试方法。本章中,陈宏刚博士主要结合自己在微软公司多年的测试工作总结出宝贵的经验,为读者介绍如何撰写测试的相关文档,包括测试计划文档、测试规范文档、测试案例文档、测试报告文档以及 Bug 报告文档。本章附件中包含了陈宏刚博士亲手撰写的一些测试文档实例。

10.1	测试计划 (Test Plan)	293
10.2	测试规范 (Test Specification)	298
10.3	测试案例 (Test Case)	301
10.4	测试报告 (Test Report)	301
10.5	Bug 报告 (Bug Report)	302
附录 A	微软亚洲研究院介绍	343
附录 B	课程设计	347
编辑手记		351

导 读

Guide

《软件开发的科学与艺术》成书的原因与过程

微软亚洲研究院成立三年多来，不但在学术领域取得了令人瞩目的成就，在促进亚太地区学术交流、推动先进学术思想在本地区传播、帮助优秀的软件技术在本地区的开发和应用方面倾注了大量心血，进行了许多不凡的工作。在中国，我们一直把支持教育、帮助中国培养高水平的计算机人才视为重要工作之一。

《软件开发的科学与艺术》一书取材于 2000 年秋季在北京大学所开设的一门选修课。当时，北京大学计算机系主任李小明老师向我们建议，希望我们能在北京大学讲一讲“微软的软件是怎样写的，微软的研究是怎样做的？”。

在国内工作三年多来，有机会与众多教育界及产业界人士探讨一些计算机产业发展的问题，我们都强烈意识到：中国学生所需要的不仅仅是一些编程的技巧，更需要一些程序设计、项目管理和从事研究的知识和经验。在这一考虑下，我们邀请微软总部以及微软亚洲研究院的多位资深研究人员、程序开发人员和项目经理在北大做了题为《软件开发的科学与艺术》（**The Science And Art of Software Development**）的系列讲座，并为此讲座编写了专门的课程设计作为实践环节，希望学生们能够从我们的亲身体会以及实践经验中学习一些实实在在的东西。

讲座异乎寻常的成功。未能参加讲座的学生和老师们再三呼吁我们将讲座的内容编辑成册。强烈的责任感和使命感让我们认识到，这是一项非常有意义的工作。不管需要投入多大的心力和资源，我



导

读

·

·





们也要成就此书，让更多的高校师生和软件业从业人员分享到微软公司多年来积累的软件研发和管理经验。

《软件开发的科学与艺术》于 2001 年 5 月正式开始整理和编辑。在长达一年的整理和编辑过程中，正值中国政府发布鼓励软件产业政策，在全国 35 所高校试办示范性软件学院。许多老师在读完初稿后认为此书非常适合做软件学院的教材或参考书。于是，我们在北大课程的基础上进行了大规模的内容扩充，加入了全球软件产业发展趋势、中国软件产业发展机会与挑战、微软二十几年发展过程中总结并形成的软件产品开发和项目管理经验、现代软件开发对人才的要求以及微软产品开发团队的详细介绍的章节。本书还用大量、详尽的案例揭示了在软件开发过程中可能遇到的问题和解决的方法。

《软件开发的科学与艺术》一书是集体智慧和共同努力的结晶：我很高兴能将近年来我对中国软件产业发展的一些看法和大家一起分享。凌小宁博士从回到中国帮助创办微软中国研究院到调回微软总部担任项目经理，一直热切关注着中国软件业发展的状况。此次，他将多年来在软件设计、软件开发和项目管理方面的经验写到了书中。张益肇博士结合自己在美国成功创办语音门户 Nuance 的经验，介绍了有关将研究成果转化成产品的经验。微软亚洲研究院新技术开发部的林斌结合自己在微软及其他美国公司多年的软件开发经验，总结了写好代码的秘诀；陈宏刚博士结合自己在微软多年的软件测试经验，系统地总结于软件测试的过程和方法；熊明华在微软的产品部门任项目经理多年，当他得知此书的写作消息后，在繁忙的工作之余，总结于自己对项目管理的经验和体会，并对书稿进行了多次认真的修改。

感谢本书编写组织委员会的成员，包括微软公司马歆、陈宏刚，电子工业出版社郭立，是他们的辛勤劳动推动了本书的顺利面世。

感谢俞俊平和余安萍，他们在数月间认真的整理素材才使得本书的面世成为可能，本书的字里行间都凝聚了两位整理者的心血。

感谢微软公司的翟力红小姐、郝海洋先生、彭云峰先生，他们为本书的部分章节提供于部分文档和案例，增强了本书的可读性和实际应用性。

《软件开发的科学与艺术》的内容

软件开发是一门科学，更是一门艺术。如何学习和借鉴微软公司在软件开发方面的成功经验，了解它独特的企业文化，对于中国软件企业的未来发展可能有所裨益。

《软件开发的科学与艺术》一书将收录以下内容：

1. The Challenges and Opportunities of the Global Software Industry

全球软件产业现状、趋势与挑战

2. Talent Beyond Technology -- What Kind of Talent We Need for Modern Software Development

现代软件开发对人才的要求

3. From Research to Products

从研究到产品

4. Software Development at Microsoft

微软的软件开发

5. Source of the Software Design

软件设计之源

6. Program Management

项目管理

7. Things you can Do to Write Better Code

写好代码的十个秘诀

8. Secrets of Software Performance

如何提高程序的性能

9. Basic of Software Testing

软件测试基础

10. Write Good Testing Documents

如何撰写测试文档



本书的意义

具有极高的实践性和实用价值是本书的主要特点。通过阅读本书，我们希望大家：

- ✧ 学会如何在现代 IT 企业的文化环境中做一个成功者
- ✧ 学会如何做世界级的、高质量的研究
- ✧ 学会如何创建大规模的软件产品

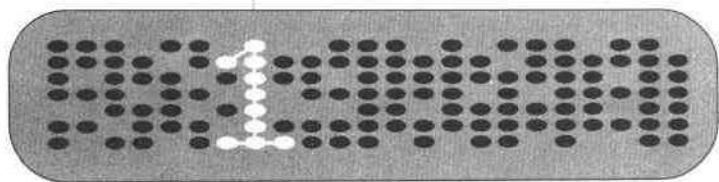
如果你想：

- ✧ 成为一个成功的工程师
- ✧ 成为一个成功的研究人员
- ✧ 成为一个成功的管理人员
- ✧ 为一个成功的软件公司工作

你会发现这本书的内容对你是非常有帮助的。

《软件开发的科学与艺术》作为第一本由微软员工编写、介绍微软公司软件研发经验的中文书籍，将成为中国软件教育和软件企业很好的参考资料。我想借此机会代表所有在微软工作的中国工程师，表达对中国软件产业健康发展的诚挚祝福；表达微软亚洲研究院对支持中国软件教育事业的诚意和长期的承诺。我们将进一步通过教师培训、课程建设、对软件学院的支持等多种形式，为中国的软件事业发展尽一份力。

微软亚洲研究院院长、首席科学家



第 1 章

全球软件产业现状、趋势与挑战

The Challenges and Opportunities of the Global Software Industry

背 景

软件是一台计算设备的思维中枢。经过数十年的发展，软件产业已经成为当今世界投资回报比最高的产业之一，而这一产业正在潜移默化地改变着我们赖以生存的这个星球的面貌。新世纪，软件产业已经呈现出了一些引人入胜的转变迹象，本章概括地把这些迹象表述为三个趋势：网络化、服务化与全球化，并从技术角度就高科技产业所面临的机遇与挑战阐述了作者的观点。

微软亚洲研究院院长兼首席科学家。数字影像和视频技术、多媒体通信及Internet方面的世界级专家。

1966年 出生于山西太原。

1978年 年仅12岁时考入中国科技大学少年班。

1989年 获得美国乔治·华盛顿大学电气工程博士学位。

曾就读哈佛大学高级主管商业培训课程。

1997年 年仅31岁时被授予美国电气与电子工程师协会院士(Fellow of IEEE)称号,成为该协会100年历史上获得这一荣誉的最年轻的科学家。

1998年 获得美国电子工程师荣誉学会授予的“杰出青年电子工程师奖”。此奖项是美国工程界的权威性大奖,每年授予一位在电子工程学方面做出杰出贡献的35岁以下青年科学家。

张博士拥有50项美国专利,其中多项专利已成为ISO和ITU的标准,被许可给许多公司使用。此外,他与同事合作出版了《图像和电视压缩技术》等11本专业著作,并先后在世界权威专业杂志上发表了200余篇有关影像编码和处理、多媒体无线通信、互联网视频、卫星通信、数据网络、医疗影像和通信方面的论文。目前,他除兼任多种权威学术杂志的主编或编辑外,还兼任多家高科技公司董事及政府部门的科技顾问。



张亚勤院长在回答同学的问题

张亚勤

本章内容概览

- ☞ 软件产业的网络化趋势
- ☞ 软件产业的服务化趋势
- ☞ 软件产业的全球化趋势
- ☞ 网络化、服务化、全球化趋势对中国软件产业的启示

创新的动力源于人类不断升级的需求和希望。计算机的发明标志着信息革命的开始，然而，20 世纪 40 年代的人们无论如何也不会预见到，未来的世界会因为一台名叫 ENIAC 的庞然大物而发生翻天覆地的变化——1946 年，这台结合了当时最先进的雷达脉冲技术、核物理电子计数技术和通信技术的庞然大物标志着人类有史以来最重要的计算工具的诞生。

然而，真正改变了普通人生活和工作面貌的是 20 世纪 70 年代出现的微型计算机。从最初的 4 位单片机到现今功能卓越的吉赫主频个人电脑，其功能演进的速度可以用“令人瞠目”来形容，而其影响也逐渐波及到更广泛的社会经济领域中——计算机硬件尤其是微处理器日新月异的更新速度牵动了全新运算体系的发展，同时硬件系统对相应软件也提出了愈来愈严格的要求。

软件是一台计算设备的思维中枢，经过数十年的发展，软件产业已经成为当今世界投资回报比最高的产业之一，而这一产业正在潜移默化地改变着我们赖以生存的这个星球的面貌。新世纪的软件产业已经呈现出了一些引人入胜的转变迹象，在此我可以概括地把这些迹象表述为三个趋势：网络化、服务化与全球化。



1.1 软件产业的网络化趋势

NASDAQ（纳斯达克）——曾经塑造了像微软（Microsoft）、英特尔（Intel）、思科（Cisco）等耀眼的明星而被誉为孕育高科技梦想的温床。然而随着网络股戏剧性地急涨和陨落——NASDAQ 指数仅一年之内就从最高 5 300 点跌到最低 1 600 点，梦想变成了不断破碎的泡沫——经营不善、资金短缺、模式落伍……一家又一家新兴的.com 企业或一蹶不振，或改弦更张，残酷的现实使旁观者们更加担忧：网络经济有前途吗？人们产生这样的疑问是完全可以理解的。然而，在身处网络革命暴风雨中的探索者看来，.com 泡沫的破裂未必是坏事，相反地，它会使投资者、华尔街乃至整个 IT 产业都将以更加成熟和理性的目光审视新的经济模式的发展轨迹。微软公司总裁史蒂夫·巴尔默先生曾这样评价：“.com 泡沫的破裂对于整个 IT 产业来讲是再好不过的事了。华尔街和创业者现在应该明白了技术的重要性及公司的真正价值。”某些.com 的陨落不过是不够成熟的网络经济走向衰落的前兆，而真正的网络经济才刚刚崛起。目前有两种趋势：一方面许多小型的.com 公司正在走向低势或破产；另一方面，主流公司正加大在网络上的投资。例如，微软的.NET 战略将把整个公司的发展架构在网络之上。未来的软件和信息服 务都将建立在网络服务的基础之上。

计算与通信的融合趋势是不可逆转的，人类的生存方式、企业的经营模式正在被这种趋势更新和改善。时间与空间造成的沟通障碍正在逐渐消失，沟通将能在任一时刻、任一地点、通过任一手段实现。

回想一下，我们大致可以将网络的发展划分为以下三个阶段。

（1）起始阶段（20 世纪 70 年代至 90 年代）——某种程度上，二十世纪六七十年代的美苏冷战加速了网络的诞生：从 ARPANET



(1969 年)到 NSFNET,正是在军方的支持下,科学家们开始了这项注定会对亿万人的生活产生影响的伟大研究。对相关硬件的研发很快取得了成果,顺理成章地,人们需要制定一个可作用于所有计算机的传输协议,于是,TCP/IP 协议在 1974 年 5 月发布——这也成为网络发展史上的里程碑。尽管连接电脑并在机器之间实现信息有效传输的实验在一定程度上取得了成功,但受益者的范围只局限于学术界和研究界的一小部分人中。记得在我刚到美国时,曾与一位学友甚是投机。1986 年初的一天,我邀请他一起用餐,他回答说要先给一个朋友发送信息——我看着他坐在计算机前面敲敲打打,只几分钟信息即发送完毕。就在我一头雾水的时候,这位学友告诉我,他刚通过计算机成功地发送了信息。从那时起,我才知道世界上有 E-mail 这种便利的沟通工具。在这一阶段,Internet 的功能主要是通过文档和电子邮件的传输来实现人们所需的物理上的连接。

(2) 万维网 (World Wide Web, WWW) 阶段 (20 世纪 90 年代到今天): 1991 年,一位欧洲软件工程师 (Timothy Berners Lee) 发明了万维网。不久,网景公司推出了 Netscape Navigator 浏览器,微软公司也在其后发布了 Internet Explorer,浏览器软件使得 Internet 不再只是传送信息的平台,而进化成为呈现信息的窗口。简单的 HTML 工具、生动的浏览器和丰富的内容使 Internet 迅速地成为了人们日常生活中不可或缺的一部分。时至今日,网络已经不仅仅是某种令人震撼的技术成果,它已经演变成人们进行创造和交流的广阔舞台。然而,这一阶段的 Internet 仍然只是一个呈现信息、供人们浏览的静态的平台。

(3) 智能网络 (Intelligent Web) 阶段 (现在到未来): 目前,我们正在进入一个网络技术发展的新纪元,网络技术正呈现出四方面的变化趋势:从静态网到动态网,从被动方式到主动方式,从呈现信息和浏览的窗口到智能生成的平台,从 HTML 到 XML。互动性和可编程性成为崭新的动态网的主要特征。

不难看出,软件技术的革命推动了网络技术从第二阶段到第三



阶段的飞跃。事实上，软件在网络技术中的作用，像半导体、光纤在通信发展过程中的作用一样，扮演着越来越重要的角色。

技术的创新和发展将使网络应用者改变先前以不变应万变、被动地处理信息的状态，并以更加灵活主动的姿态去面对眼前的虚拟世界：整合了服务器、路由器、转换器的软件服务的价值将在网络用户端实现最大化，软件将会成为网络发展和应用的最重要的动力：著名的“摩尔定律”（Moore's Law）、“贝尔定律”（Bell's Law）、“吉尔德定律”（Gilder's Law）和“梅特卡夫定律”（Metcalf's Law）成为网络时代的基本定律。相关硬件的性能将越来越高，无论是企业或是个人都将得益于无限的计算和充裕的带宽，并由此促使网络的价值急剧膨胀。包括 CPU、内存、图形卡、带宽等在内的硬件能力空前扩展，将直接冲击软件产业的既有发展模式：凝聚了网络通信、计算机和娱乐等三个传统行业优势的新技术具有极其广阔的应用前景——互动的、动态的多媒体技术将具有个性化、结构化的智能特征及可搜索性的特征，能够依托于网络环境创建出更加生动逼真的 2D 与 3D 场景；还可以应用各种类型的智能设备。例如，人们可以把办公室和娱乐工具集合在 Pocket PC 上，并可在世界任一角落与千里之外的同行在实时视频会议上讨论市场策略，或是交流欣赏高品质数字电影的心得。新一代用户界面（UI）技术与人工智能（Intelligent Agent）技术将仅文化心理迥异、文化程度不同的普通人切实感到生活质量的飞速提升——“人机对话”正在变成看得见、摸得着的现实，而被多数人认为是“有能力思考的”计算机将有可能因为新一代用户界面技术与人工智能技术的重大突破而在某些特定领域诞生，网络化、人性化、个性化的软件将使技术不再成为人们沟通的障碍。



1.2 软件产业的服务化趋势

在网络环境下，软件研究、开发、测试和经营的传统模式正在发生改变。软件的服务化将成为一种趋势——目前的软件开发方式可以称之为“打包式”：企业首先结合用户和市场的需求，对软件的整体架构、功能设置进行定义，然后进行开发和成品测试，之后的一段时间里，技术支持部门将根据用户的反馈及方方面面的意见对软件的功能进行调整和完善；直至一两年后推出新的软件版本——厂商从这样一个“研究—发布—释放”新版的循环中获利。然而，网络革命所带来的服务化趋势为软件产业开辟了成本更低、效率更高的新的获利途径，同时使用户能够获得更加简捷、更加全面的服务享受——由于在系统后台借助网络运行，使得软件的安装与升级成为完全透明的过程，不必费心安装光盘或担心软盘损坏，不必自己找寻软件的后续版本。不过，软件存在 Bug（错误）是在所难免的，以前厂商为用户提供的解决方式大抵是推出新版本或是把相应的补丁文件放置在网站上供有需要的人士下载；而在软件服务化成为企业的普遍选择之后，服务的提供者可能早已准备好了修复的工具，而网络化、智能化的软件也许已经在你的不知不觉中自动完成了你所需要的升级工作；此外，每周 7×24 小时的实时用户服务也最大限度地解除了软件服务购买者的后顾之忧——对用户来说，软件服务化就意味着更加完善的功能、更加低廉的价格和更高品质的服务；对企业来说，软件服务化则意味着更有效的成本控制、更快捷的市场响应速度，以及更可观的利润回报。

一直以来，优质软件的价格也往往居高不下。服务化软件将为解决这一问题提供某些思路。用户有权选择不同的回报方式，可以购买软件，也可以接受广告信息，从而得到低价甚至免费的服务。

由销售包装产品到销售服务内容，我们可以发现软件产业正在



向订购服务的方向转变。事实上，目前传统的订购服务模式已为大众所接受，电力公司、自来水公司、电信公司都在以类似的形式向用户提供若不同内容的服务，而且，这种服务模式正向软件产业渗透和扩展。

我个人预计，在今后五年内，“打包式”软件将继续呈上升势头并趋于饱和，之后开始下滑；而“服务式”软件也将同时为用户逐渐接受，并迎来自己的上升期——但这并不是说“打包式”软件将最终在市场上消失，只是“服务式”软件的响应速度更快，给用户带来的利益也更多。

但现在看来仍存在一些阻碍软件服务化趋势扩散的因素。例如，新的趋势要求企业尽可能地缩短产品的研发周期——打包时代厂商推出新产品的周期大约是一至两年，而在服务时代可能就要求一至两个月，这就要求企业必须拥有效率更高的开发工具、开发方式和开发流程；其次，打包时代软件的测试过程大致是在模块开发完成之后，对组成整个系统的各个模块分别进行测试，集成测试是最后阶段的工作——在服务时代，这种大规模集成测试几乎不可能，服务提供者需要建立完善的在线监控体系，发现某一模块的 Bug 之后立即寻找解决方案，从而为用户随时提供强大的质量保障；再次，服务时代要求软件厂商的产品的可扩展性和自适应能力更强，因为网络服务的对象可能以千万计，这就不能像打包时代人们通常做的那样仅仅把产品需要的配置表罗列在包装盒或说明书上；网络本身所存在的带宽问题和延时问题都可能使不同的用户获得的服务内容产生差别。如何保证 100 人乃至 100 万人都享受同等质量的服务？解决之道惟有全面提升技术的可扩展和自适应性能；最后，由于网络革命赋予应用者的空前自由使人们对个性化服务的需求更趋强烈，相应地，软件服务提供商需要对软件的整体架构、重复使用性和模块特殊性重新优化——不同的用户对软件系统模块功能的需求也不同，在设计时，研发人员就必须考虑到这一问题，从而有意识地加强软件管理和控制性能，并保持设计的一致性和连贯性。



1.3 软件产业的全球化趋势

从中国的内陆到美国的海滨，从人头攒动的繁华都市到人迹罕至的蛮荒之地，网络的影响力已经无处不在，无孔不入。对于软件厂商而言，其所提供的服务必须能够适应不同的地域，作用于不同类别的平台，适用于各种设备，并支持不同的语言。蔓延的互联网所带来的通信与计算的融合趋势只会使一些原本便为多数参与竞争者所共同遵循的标准更加畅通无阻。

过去在计算机技术领域有许多封闭的技术和标准，有一定实力的厂商总希望把自己研发的技术作为企业的机密——如果说在单机时代，由于大家还不知道该借助何种工具实现彼此沟通，因此这种做法在理论上还有可能成功的话，那么在网络革命中，在强调开放性、交互性的时代，这种厂商的成功概率几乎为零。试图闭门造车且抵制国际通行标准的行为是没有前途的，因为确定一个为全球多数厂商以及数以亿计的用户所共同遵守的标准，是一个严肃且重要的问题。国际多数知名 IT 企业支持主流的开放式标准，反对任何封闭甚至保密的规则；其次，能够在网络世纪发挥持续指导作用的技术必须是前瞻性的，必须在未来的十余年乃至数十年内仍具备蓬勃旺盛的生命力；还有，这种技术标准应当是为工业界的伙伴企业所广泛认可和坚决支持的。我可以在这里举几个实际例子——多媒体表述方面如 ISO, MPEG 系列标准；互联网方面如 XML, Internet, World Wide Web, 如 IETF, TCP/IP 等；通信方面如 2.5G/3G/4G, 如蓝牙, IEEE 802.11 等——这些标准因其覆盖面广且影响力强，也成为微软在进行软件研发时所遵循的标准。

人才的全球化同样是软件产业全球化的一个重要特征。仿佛音乐、绘画、建筑等艺术形式的推广从不局限于任何人为界限一样，核心技术本身的意义也早已超越了政治与文化的范畴。地球





上的每一位居民都有权利享受瞬息万变的技术创新所带来的工作与生活便利。正是由于技术正在逐渐消弭国家、种族以及文化的森严界限，才使得这个时代的人才开始更多地考虑自身的发展空间和发展机遇——就像我们所看到的一样，微软公司在中国设立的研究院正在以开放式的创新研究探讨着造福于所有人（包括中国人在内）的可行途径；同样地，也有一些中国企业在美国硅谷、在世界其他地区设立科研机构或是投资办厂——这些事实都表明网络革命中人才在追随机遇，而企业则在追随资源。“国家的公司”正逐渐为“公司的国家”所取代，而技术的受益者们今后将更多地考虑“谁能使我的效率和利益最大化”，而把其他因素抛诸脑后。另一方面，技术的全球化浪潮并不会导致产品单一化，恰恰相反，基于同一核心技术的不同产品，将使不同需求、不同品位的消费者拥有更广泛的选择空间。

企业的管理模式也在经受着技术全球化的冲击。以前，一家公司的管理体系大致呈树形或金字塔形分布。而今后，这种体系将更多地向矩阵状发展——我姑且把这种模式称做好莱坞模式。大家应该非常熟悉好莱坞电影的制作流程：首先是找编剧写剧本，然后聘请导演、挑选演员，影片拍摄完成之后，由市场推广人员负责宣传造势——这是一整套合作团队，但成员之间的合作很少有固定的时候，往往是电影拍完，成员也各奔东西。之后呢，又一部新影片进入规划阶段，于是，又一支新团队诞生了……全球化趋势和先前我所说的服务化趋势使软件产业的管理模式也向着好莱坞模式靠近，令人目眩神驰的高速竞争使企业只能选择以更迅捷的速度发展，否则便可能落后于竞争对手，落伍于所处时代。在这样的形势下，软件企业亟需轻装上阵，这就要求企业在危机来临前重视自己的核心技术，尽可能地把非盈利核心的业务外包出去，让这些业务成为其他企业的盈利焦点。

1.4 网络化、服务化、全球化趋势对中国软件产业的启示

目前来看，中国软件产业正处于关键的转型阶段。经过几十年的发展，尽管中国软件产业在某些方面取得了长足的进步，但总体形势还是不能让人过分乐观。根据资料显示，目前中国各类软件企业一年的产品销售总和仅仅相当于一家国际大型软件公司同期销售的 1/10，这个令人尴尬的数字充分表明，这一产业还远未走上快速发展的轨道，其业绩与国内如火如荼的高科技产业发展状态不相适应。事实上，软件产业在中国的发展有不少得天独厚的优势。首先，相对于汽车、能源、IC 等需巨资注入才能产生效益的产业而言，软件企业所需投资相对较少；其次，软件企业赖以成长的最关键因素是脑力资源，而中国又具有显而易见的人才优势；其三，软件发展的周期较短，与传统工业相比，软件产业的短程及中程回报更加明显；其四，这一产业的高利润特征使其历来为投资和融资者所青睐，相信通过引导和管理，软件产业会成为中国经济新的投资热点；最后，中国有着巨大的内需市场，甚至更加需要并且完全可以完全消化本地化的软件产品，这也为软件产业的蓬勃发展提供了最肥沃的土壤。

然而，我们也必须正视软件产业发展过程中所遇到的阻碍。例如，如何才能创造出一个适合软件企业生存和发展的高质量生态环境——污染和毒害着这种发展环境的最主要的因素便是屡禁不绝的盗版。必须指出，如果不能扼制猖獗盗版，将最终扼杀软件产业。国内一直在努力完善保护知识产权方面的立法，同时在执法方面也不时有一些让人拍手称快的行动，但由于中国的盗版情况相当复杂，所以有必要进一步向社会各界宣传尊重知识产权、打击盗版的重要意义。这样做不仅可以保护国外厂商的投资热情，更可以保障国内



软件企业的合法权益。至于软件企业的发展模式，一般是先寻求投资，之后展开研发，接着把产品推向市场，由此获得利润并最终为上市做好必要准备，只有实现了整条产业链的闭合才能实现整个产业的良性循环和发展。而在中国，由于盗版原因，守法的软件企业投入巨大的人力、物力与财力，回报却少得可怜。存有漏洞的产业链仍然呈发散式状态，加上金融上市机制本身又不尽完善，以上问题自然会使整个产业看上去缺乏持续发展的动力。

软件产业持续健康发展需要政府的大力扶持——在基础设施建设方面如此，在企业生存环境的建设方面亦如此，产业的发展成熟必须依赖于市场机制和法制的完善。

改善人才的知识结构和管理水平也同样重要。智慧在华人的头脑里，中国并不缺少天资聪慧的优秀人才，然而目前在软件研发过程中，有的人虽然能够完成某一模块的研究任务，却不能胜任系统方面软件的研究工作。另外，既通晓技术，又懂得管理的人才相对较少。在我看来，现在中国 IT 企业并不缺少 CEO，也不乏程序员，惟独缺乏对产品发展机制相当熟稔的“架构师”，以及对市场和技术具有前瞻能力的 CTO——首席技术执行官。

软件人员的素质直接影响企业的发展和前途。现在国内高校的课程机构设置不尽合理，课本内容也颇有过时老化的迹象——不应该让我们的人才因为这些客观因素而落后于时代；此外，职业道德教育应该作为相当重要的教育内容刻在每个学子的脑海里——毕竟，“有大德才有大智”。

新世纪里，计算与通信技术的应用者们在这个由网络编织的广阔世界里，头脑中是与阿基米德的预言几乎同样伟大的梦想：给我一个接口，我就能驱动地球——互联网的急速蔓延在理论上肯定了这句话所蕴含的意义。在可以预见的未来，每个人都会发觉自己需要一个与信息新世界相连的接口，而锻造这一接口的，便是网络化、服务化、全球化趋势推动之下的软件产业。在未来的 20 年，软件业将成为全球高科技产业发展的最主要的推动力。这为发展中的中国



也带来了前所未有的机遇和挑战。我坚信，中国软件市场的发展将令全球软件企业为之瞩目，而中国的软件产业也必然能够把握住前所未有的机遇，在不久的将来傲然崛起。

第 1 章

1

全球软件产业现状、趋势与挑战

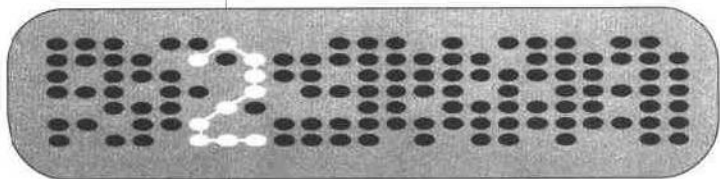
第2章

现代软件开发对人才的要求

Talent Beyond Technology—What Kind of Talent We Need
for Modern Software Development

背 景

软件开发是一门科学,更是一门艺术。微软在二十几年的发展过程中形成了其独特的软件开发与设计的企业文化。目前,中国软件业的发展喜忧参半。中国拥有高素质、基础扎实、学习能力强且思维敏锐的软件专业人员,但中国大规模的软件生产尚处于初期发展阶段,软件的研究与开发过程中尚有许多亟待解决的问题。本章中,凌小宁博士关于微软企业文化以及软件开发人才等方面作了精辟的论述,这些内容是我们中国学生及软件从业人员应该了解、理解并恰到好处地遵循的。



凌小宁

1993年加盟微软，曾在多个产品和研究部门担任软件设计工程师、软件开发组组长、软件开发部经理、软件开发总工程师和项目管理经理。在图形学、多媒体及用户界面方面获多项美国技术专利，并发表过多篇计算机人工智能和数据库方面的学术文章。

1982年曾获北京大学计算机硕士学位，1990年俄勒冈州大学计算机博士学位。

凌小宁博士在美国计算机工业界工作的12年中，积累了丰富的软件产品战略策略规划、产品设计及产品研究开发的经验。



凌小宁博士在微软总部的办公室里

本章内容概览

- ① 现代软件开发概述
 - 独具魅力的微软企业文化与软件开发人员的培养
 - 从差别中寻找解决方案
 - 扎实的基础和创新、独立的工作能力
 - 主人翁精神和团队精神
 - 锲而不舍、从错误中学习的精神

2.1 现代软件开发概述

2.1.1 引言

微软公司在 1975 年时只有 3 名员工，营业额仅 16 000 美元；到 1989 年时已经有 8 000 名员工，营业额达 80 亿美元；而发展至 2000 年时员工已多达 35 000 名，营业额达 240 亿美元，获利更高达 150 亿美元，成为世界上最大的软件公司。这一发展过程堪称世界软件业奇迹之首。微软的产品分类细致、辐射面广、人性化设计独到、竞争力强、使用覆盖率高；微软的企业文化在众多 IT 企业中独树一帜且极具亲和力，平等、自由、随和、创新的理念构筑了公司内融洽的沟通环境，微软员工的主观能动性和创新精神被最大可能地触发；微软的员工是积极又绝顶聪明的，团结协作、创新独立、锲而不舍的精神在微软的员工中体现得淋漓尽致。



2.1.2 传统软件开发与现代软件开发

传统软件开发又被称为作坊式的软件生产。顾名思义，开发工作主要依赖于开发人员的个人素质和程序设计技巧。其特点是，缺少与程序有关的文档，软件开发的实际成本和进度与预计的相差甚远。由于程序量和规模不大，通常都由单人编写，不需要考虑团队合作，所以项目的管理松散，程序可重用的程度差。同时，由于项目成败系于开发人员一身，因此失败的风险增加，可维护性差。随着计算机应用需求的快速增长，软件规模也越来越大，然而软件开发的生产率，远远跟不上应用需求的增长速度，20 世纪 60 年代中期，人们把在软件开发和维护中的各种问题，称为“软件危机”。在 1968 年德国 NATO（北大西洋公约组织）会议上，引入了现代软件开发的方法，希望用工程化的原则和方法来克服危机。

现代软件开发适应了社会化大生产的要求，强调采用分工和协作，重视对项目的管理和软件质量的把握，采用了工程化的方法进行文档的控制和代码的管理，不再像传统软件开发那样，从设计到开发到测试都是一人完成，这就有效地保证了软件的质量。

微软在现代软件开发（计划、设计、实施、测试乃至市场运作等）方面有着一定的权威。图 2.1 所示就是微软软件开发的典型体系结构。

在微软的软件开发模式中，开发人员的职责明确，分工细致。从前瞻性的技术研究到研究成果转向产品，再从产品规划到项目经理的产品设计，以至到各个项目开发团队，最后到测试人员，都有明确的分工。微软的软件开发有许多独到之处，其中组织完整的软件测试团队是软件生产过程质量的过滤网。素质优秀的软件测试团队可将软件开发过程中的维护费用降至最低，同时又提升了软件质量以及公司信誉。而所有这些又都孕育在微软成功的企业文化之中。

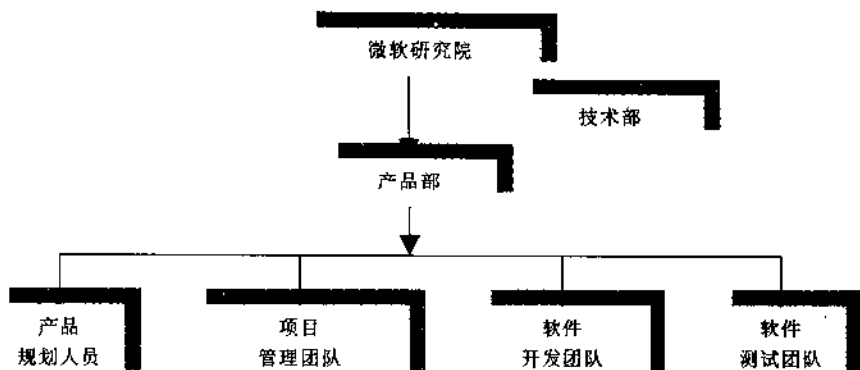


图 2.1 微软软件开发的典型体系结构

现代软件开发模式有着巨大的转型，如单枪匹马写出 WPS 的求伯君、单独完成 BASIC 的比尔·盖茨和保罗·阿伦（Paul Allen）那样的创业程序英雄，在现代软件开发中会越来越少了。越来越多的项目经理，将不仅仅只是计算机高手，许多人同时还是应用领域的专家，或者具有丰富的管理经验。而项目的划分，也将会越来越细，项目不再是依赖于单个程序员的发挥和技巧，依靠的是团队（Teamwork）的力量。

那么，在这样的现代软件开发模式下，到底要求我们具备有什么样的素质和能力呢？我们又该如何去做，才能适应未来的发展呢？

2.2 独具魅力的微软企业文化与软件开发人员的培养

企业文化实际上指的是一种企业环境，它反映了企业的员工在这个环境中是如何工作的。毋庸置疑，一个软件公司的企业文化对于软件开发有着很重要的影响。一个企业文化营造得好的企业，无



疑能够更大程度地发挥员工的潜能，激励员工的创新，加大员工的沟通，增强企业内部的凝聚力，使员工在充分协作的环境中工作。这样的工作“软环境”会极大地吸引多方面的优秀人才。

我并不认为微软的企业文化在所有方面都是优秀的。实际上，我个人认为，在某些方面，微软的企业文化也存在许多问题。但是，微软能取得今天的成就，其企业文化中自然有其成功的一面。在这里我主要将自己认为对中国 IT 业有用的微软企业文化提取出来，向大家作一介绍。事实上，微软的企业文化也代表了美国绝大多数成功 IT 业的企业文化。

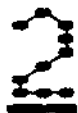
我们很难评判一种企业文化的对与错，每一种文化都有它存在的原因。只是目前着眼于现代 IT 业的发展情况，我认为微软和美国的企业文化更能够适应现代 IT 业快速发展的节奏，更能够满足现代 IT 业快速发展的需要。

企业“以人为本”，企业中的人则应积极热情地投入自己的工作。在微软，随时可听到或看见“Where do you want to go today?”（今天你到哪儿去？）这句友善且热情的话语。在微软每个人都需要拥有“激情（Passion）”。

概括微软企业文化宗旨的一句话：

“Wake up every day with a feeling of passion for the different technology will make in people's life.（每天醒来的时候要对技术给生活造成的改变始终拥有一份激情。）”

几乎所有的微软人都能够说出上面的这句话。这句话里面有两个关键字需要注意。一个是激情（Passion）。如果你每天早上醒来的时候都有这样一种感觉：“我就想躺在床上，我不想去上班。”这就意味着这样的一种征兆：你对目前所从事的工作或项目缺乏兴趣，



已经没有激情了。这个时候你就要考虑换一个工作或项目去做了。因为工作如果不能够吸引和驱动员工本身，不能够给其带来工作动力，那么，员工就不会完全投入地工作，自然就无法取得高效率，企业也无法获得高的生产力。相反，我们应该拥有这样的一种激情：每天早上醒来的时候都有一种强烈的兴奋感，想着要赶快把自己手中非常具有挑战性的项目做完。

另一个关键字是技术（technology）。你要乐于向不同的技术进行挑战，并愿意看到自己的技术被成千上万的人使用，然后尽情享受成功给自己带来的乐趣。

微软的飞速发展是世界 IT 界公认的奇迹，其中最重要的原因就是微软拥有世界上一流的人才。在传统的行业中，一个优秀的人才，可能有几倍于一个普通人才的生产效率。而对于现代的 IT 业而言，一个优秀人才的生产效率，可能是一个普通人才的数百倍！一个好的程序员与一个一般的程序员之间的差别是非常非常大的，有时候一个水平较差的程序员甚至会起到相反的作用。这一点和软件工程的一些统计数据也是相一致的：在影响软件产品质量的众多因素中，占最主要地位的是开发人员的生产效率。

要留住优秀人才需要有良好的企业文化，吸引优秀人才更是这样，因为每个渴望成功的人都希望加入一个能重视自己、能让自己充分发挥能力、能提高自身能力的“大家庭”。

“以人为本”，是微软公司文化环境中最重要的方面。作为一个成功的企业，微软拥有高素质的软件开发人员。他们懂得如何理解他人的工作，如何将自己与他人的工作整合起来，如何剪裁他人的工作。他们会合理地安排时间进程，及时总结工作中的成就和失败，以高效为目标，而非以经常超时工作为荣。

比尔·盖茨认为，微软最宝贵的财富不是巨额资产，而是“人”。正是众多的人才创造出了巨额资产。

在微软流传着这样一个故事

有一次比尔·盖茨和上帝谈话。上帝对他说：“地球明天就要毁灭了！由于你是如此的成功，因此我特许你从地球上带一样东西到天堂！你想带什么东西？”比尔·盖茨是这样回答的：“请允许我从微软选 300 个最优秀的人，我要将他们带到天堂！”

还有一件事是我自己亲身经历的。

一件亲身经历的事

比尔·盖茨除了微软公司以外，还有一个私人的软件公司。当时我在美国大学毕业以后就在比尔·盖茨那家私人公司从事多媒体方面的研究。后来微软的一些工程师发现我们做的研究与微软正在做的项目非常相似。于是他们就向微软的董事会报告：比尔·盖茨公司做的项目正在与微软的项目竞争。于是董事会就对比尔·盖茨说：“比尔，对不起，按照规定，你自己公司的产品不能够和微软的产品进行竞争！”其实，比尔·盖茨的本意并不是要和微软进行竞争，因为那是没有任何意义的，这只是正好巧合罢了。于是比尔·盖茨就说：“那这样吧！我把自己公司的技术转让给微软。”微软董事会同意了。这一笔交易比尔·盖茨赔了一笔钱，但他并不在乎。

就在进行技术转让的前一天晚上，比尔·盖茨找我们谈话，他说：“虽然我们公司要解散了，虽然我们的项目要停止了，但是我们仍然拥有最宝贵的财富，即你们这些研究人员！”



同样，尽管微软的 Windows, Exchange, SQL, Office 等都非常重要，但是如果没有像你们这样的人才，微软也将一无所有。你们这样的人才对我们公司和微软来说绝对是处于第一位的！”

这些都说明：在比尔·盖茨的理念中，人是最重要的。

正因为如此，微软在招聘人的时候始终遵循“宁缺勿滥”的原则，力争招到最优秀的人才。微软的面试是非常严格的，持续的时间也非常长，甚至会给面试者一种很不公平的感觉。一般来说，微软在通知你去面试的时候，通常会有 4~6 个甚至更多的专家及负责人在等着对你进行面试，其中每一个人都要和你面谈一个小时。主要会集中考查你是否具备以下几个方面的能力：

- ✎ 扎实的基础
- ✎ 创新、独立的工作能力
- ✎ 主人翁精神和团队精神
- ✎ 沟通与协调能力
- ✎ 成就感强，有激情
- ✎ 自觉地干好工作
- ✎ 锲而不舍，从错误中学习

微软并不看重考试分数，更重要的是看重是否具备上述这些能力。下面我会对其中的几点进行具体讲述。但在此之前，我想让大家了解一下中、美两国人才的差别和研究方法的差别。这是我从在中国和美国多年工作经历当中总结出来的。从中大家会发现中国的人才在适应国外先进的企业文化方面还存在一定的差距，我们在看到这些差距的同时，要找到适当的解决方案。



2.3 从差别中寻找解决方案

国内计算机软件的开发和应用，无论从产品成熟度、市场效应，还是从开发管理等方面，都同国外有着相当的差距。造成这种差距的因素是多方面的，单就软件开发这种智慧密集型产业的重要生产要素“人”来说，国内从业人才的特点和从事研究的方法同国外有着本质的区别。只有认识到差别，了解差别造成的差距，知道差距的根源，并在此基础上谋求改进，才能获得更大程度上的进步。我对中、美两国人才的差别和研究方法的差别进行了仔细的观察，并做了简单的归纳，如表 2.1 和表 2.2 所示。

表 2.1 中、美人才的差别

中国人才的特点	美国人才的特点
敢冒风险	敢冒风险
有雄心壮志	有雄心壮志
能学习、适应新环境	能学习、适应新环境
实事求是的作风	创新精神
有点克服困难的毅力	如果对问题有兴趣，则有热情、有主动性
扎实的理论基础，尤其是数学	独立从事研究的能力
很强的编程能力	题目想得远、做得深
讲纪律、讲服从	对什么事都有主见
对许多事情都没有主见，即使有想法也不直说	直截了当地沟通甚至批评和争论

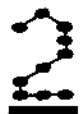


表 2.2 研究方法的差别

微软的研究方法是	微软的研究方法不是
想着做事情	坐着想事情
经过科学手段、大量的数据、可重复的深入研究	肤浅的、无用的、无法扩张的简单结果
研究、理解、借用前人的结果	不看别人的研究，或只抄袭别人的研究成果
经过产品设计工程原型，证实对用户有用	理论的、没用的纸上谈兵
承认失败，从头开始	不承认失败，永无止境地延续研究
专家带头，研究员，研究生学习	博士生带头，本科生编程

2.3.1 中、美人才的差别

从表 2.1 可以看出，在“敢冒风险”、“有雄心壮志”、“能学习、适应新环境”这三点上中、美两国的人才是一致的，但在其他方面的差别却是非常明显的。

在中国传统文化的熏陶及教育制度的培育下，“国产”人才有着实事求是的作风、克服困难的毅力、顽强的治学精神、踏踏实实的钻研精神、扎实的知识体系结构和理论基础，从事软件开发的人员具有极强的编程能力。但从另一方面来看，中国学生比较守纪律、讲服从，不去想一些出格的事情，而创新精神却正好需要不拘一格的想法；并且，中国学生对许多事情都没有主见，或者有自己的想法却不直接说出来，宁愿闷在心里，这是非常不利于团队合作的。大家平常所说的“三个诸葛亮抵不过一个臭皮匠”、“三个和尚没水喝”就是这样的道理。

美国人的创新意识却非常强，他们总是在想一些新花招，总是在想着要别出心裁，总是想方设法地要研究出一些新的东西来；但



美国人只有在对某个问题感兴趣时才会付出热情去主动研究它，否则他根本就不会花时间去研究；美国学生独立从事研究的能力也很强，对题目想得远、做得深，对任何事情都有自己的主见，并且他们敢于当面发表个人的意见，甚至进行批评和争论，以获取直截了当的沟通。

相比起来，我们缺少的是大胆的创新与尝试，缺少独立从事研究的能力和强有力的主观能动性，以及直截了当的沟通方式，而这些特点都是现代软件开发过程中处处需要的，它们能增强软件产品的竞争力、提高生产效率、降低生产成本、增加生产利润。

2.3.2 研究方法的差别

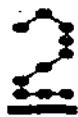
在研究方法上，中国人才和美国人才之间也有相当大的差别。表 2.2 中左边所示的是微软的研究方法，它可以代表美国人才的研究方法；而右边所示则是目前国内某些单位存在的研究方法。

有一次国内一个著名的计算机专家到微软去参观。在参观完之后，他觉得非常奇怪，对我说：“你们微软从事基础技术研究的人员怎么会大部分都坐在那里写代码呢？”他觉得既然是从事基础技术研究，就应该坐在那里研究一些理论，是“坐着想事情”。

有这样一个笑话：由于计算机（Computer）不是科学（Science），所以需要在 Computer 后面加上 Science 组成 Computer Science，使得计算机变成一门科学；而像化学（Chemistry）、物理学（Physics）等都是科学（Science），所以就不用再在 Chemistry, Physics 后面再加上 Science。

实际上，计算机是一门非常独特的科学。这门科学需要我们亲自动手去做才能取得成果。不应该在那里“坐着想事情”，而是要“想着做事情”。

计算机科学需要经过科学手段、大量的数据、可重复的深入研究，而不能靠投机取巧的手段获得一些肤浅的、无用的、无法扩张的简单结果。



案例分析

我曾经看到过一个新技术软件演示 (Demo), 这个软件的设计目的是帮助非英语国家的用户进行英文写作。当时的演示结果令我非常振奋, 因为这个软件能够非常准确地根据上下文将汉语拼音翻译成最恰当的英语。

我觉得这个软件非常有用, 于是就把微软的 Office, Word, IE 等开发团队的项目经理都找来了, 请他们看这个演示。看完之后, 他们都觉得非常惊奇, 认为这个软件非常好, 都想用到各自的产品中。于是我们就把这个软件分别拿到各自的产品中进行测试, 却发现它根本就无法正确地工作!

这是为什么呢? 原来, 在做那个演示的时候使用了一些独特的数据, 在使用这些数据中包含的例句进行测试时, 软件能够很好地工作。但是, 如果使用其他的例句进行测试时, 软件就无法正常地工作了。这个软件根本就不具备一般性和广泛适用性。因此, 可以说这种做学问的方法是很不负责任、很不科学的, 所取得的只是一些肤浅的、无用的、无法扩张的简单结果。

案例分析

还有一次, 一名副教授兴冲冲地来找我, 说他有一个非常好的想法, 他已经想了好几天并写出来了, 让我看一看。后来, 微软的一名研究人员看了以后, 说: “美国十年之前就已经研究出来了, 而且现在做的产品非常好了!”

这样的事情是经常发生的，这令我感到非常悲哀。花费大量的时间、金钱和精力去研究一项技术，在费尽千辛万苦做出来之后却发现别人早就已经做出来了。这就是没有研究、理解、借鉴前人的研究成果所造成的。

典型错误

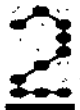
许多博士生在进入微软后，最大的困惑就是：“我为什么还要写代码？我已经博士毕业了！”

其实，在微软从事研究的人员，不论级别高低，他们在工作都离不开自己编写代码。例如，主管在美国所有四个研究院的微软公司高级副总裁里奇·雷斯特博士（Rich Rashid）仅在 2000 年就写了五万行代码，1998 年度“图灵奖”获得者吉姆·格雷博士（Jim Gray）现在仍然在写代码。研究计算机科学必须要动手做。只有动手做了，程序运行结束了，数据出来了，你才能慢慢地感觉到问题在什么地方，技巧在什么地方。如果不动手做，你就不会知道研究的结果是否正确。只是靠自己凭空想象，是永远也想不清楚的。

典型错误

国内的博士生进入微软以后另一个困惑就是：“我在学校负责整个项目。我管理一些硕士生、本科生，我负责抓总体规划，具体的编程都是由他们来完成。”

这种现象在微软从事研究的团队中是几乎不可能发生的。只有



自己亲自动手写代码，你才能真正体会到问题的所在。

由于在软件开发的整个循环链中，“研究”相当于软件开发的思维之源，其地位十分重要，因此我们应该明确从事研究工作的正确态度。做研究应建立在有实践价值的基础上，研究应具有可拓性，应能深入细致地进行，研究的成败应有足够充分的实验数据支持，而非“纸上谈兵”。

在不到十年的时间里，微软研究院（Microsoft Research）已经成为世界上最优秀的研究机构之一。在许多评比中，微软研究院已经和斯坦福研究院（Stanford Research）、贝尔研究院（Bell Research）等权威研究机构并驾齐驱了，在许多方面如微软图形研究院（Microsoft Research Graphics）甚至已经做到了最好。这与微软具有良好、高质量的研究方法是分不开的。实践证明，微软的研究方法是成功而有效的。

通过表 2.1 和表 2.2 的比较，现代软件业对我们的要求已经一目了然，这也是我们最应关注和思考的。

2.4 扎实的基础和创新、独立的工作能力

人们都知道扎实的基础对于自己来说是非常必要的。然而，在当前的社会环境下，许多人还是难以避免追求实际利益、浮躁求快的心态。很多人不重视打好扎实的基础。许多学生很聪明，很小的年纪就能写出漂亮的代码了。然而，到了后来，成就依然有限，这就是因为他们基础没有打牢的缘故。而现在的大学生，也有这样的趋势，不重视基础学科的学习，只是急功近利地想一步成功。现代的软件开发人才，不只是会编码就可以了，还需要有分析、设计和研究的能力，而这些能力，都与扎实的基础密切相关。

一般认为，中国学生的创新能力比外国的学生要差些。事实上，中国学生的思维能力不一定比他们差，而是因为国内的教育制度束

缚了学生的创新能力。所以，很多学生出国之后，照样有很多创新和发明。创新的思维对于一个软件开发人员而言是必不可少的。

案例

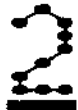
我曾经面试过一名国内一流名牌大学的博士生，他在学校的成绩是绝对的 No.1。但当我考察他的创新和独立工作能力时，他的表现非常令人失望。如果考书本上的问题，他会解释得很完美，但是就此问题进行进一步的思考和理解时，他就显得不知所措和没有头绪了。

我记得杨振宁教授曾经这样说过：“美国的教育制度是着眼于训练最好的人才，而中国的教育制度是着眼于训练一般的人才；美国的教育制度不惩罚失败，而中国的教育制度却惩罚失败。”我想，这句话应该也能说明中、美两国人才之间差别的原因吧！

面对国内的教育制度，我们应该感谢老师们辛勤的教育和培养，基础教育的成功也是非同寻常的。创造力同样可以培养，而且创造力的培养更多需要自己主观能动性的最大调度、积极热情的工作态度、良好的沟通和互相学习的技巧。这些都是我们应注意并自我调整的。

2.5 主人翁精神和团队精神

现代软件开发不再是一个人单打独斗了，软件是团队分工和合作的结晶。微软很注重培养员工的主人翁精神（Ownership）和团队精神（Teamwork）。这里，主人翁精神不是个人英雄主义或独裁，而是在有团队意识前提下的主人翁精神；同时，团队意识也



不是“一切跟着领导走”的意识，团队内的成员应以主人翁精神为出发点发挥自己的主观意识，以团队利益为己任。因此，二者相辅相成。

2.5.1 主人翁精神（Ownership）

在微软，当你说到“Ownership”（主人翁精神）的时候，就意味着你不仅拥有责任，而且拥有权力和利益。你做的事情由你自己来决定，而不是别人来决定，这就是权力。需要充分发挥个人的主观能动性。你获得权利的同时也担负了责任，一方面你应尽可能努力地工作来推动整个项目走向成功，包括计划、协作、进程安排等；另一方面应竭尽全力寻求机会和最佳的解决方案，为此甚至牺牲自己的业余时间和额外精力，即我们中国人通常理解的“责任”。你要花 200% 的力量，无论发生任何事情，你都要做好；无论遇到什么障碍，你都要克服。这就是牺牲精神。

在微软，一个大的项目通常都会分成许多很小的团队。例如，Windows 2000 项目包含了将近 3 000 名工程师，他们被分成了几百个小的团队。这样，即使一个项目再大，每一个具体的团队仍然很小，从而使得团队中的每一个成员都有一种拥有这个团队或项目的感觉，都有权力对项目的开发做出自己的决定。

另外，在微软，Ownership 还以配股权（Stock Option）的形式体现出来。微软的每一个员工都拥有配股权。对于每一个刚刚加入到微软的员工，公司都会根据他的资历、能力等情况给他配一定数量的公司股票，而且还会定期根据该员工所做出的成绩来增加或减少股票的数量。具体来说，假设某一个员工被公司配给了 1 000 股的股票，当时公司每股股票的价格是 50 美元，如果一年后公司的股票上涨到了 80 美元，那么每股的差价 30 美元就归该员工所有，这样，该员工就一共可获利 $1\,000 \times 30 = 30\,000$ 美元。这种机制的目的就是使得每一个员工有一种和公司荣辱与共的感觉。下面的事情是我亲自经历的。

案例

微软公司的内部有一个刊物叫“Microsoft News”，有一天我在该刊物上看到许多员工在讨论厕所的手纸。有人质问说：

“我发现公司厕所里的手纸每天还没有用完，第二天就被换成新的了，不知道那些没有用完的手纸都到哪里去了，这对公司是一个很大的浪费！”接着就有人解释：“之所以每天都换成新的手纸是害怕手纸用完了大家没得用，另外，没用完的手纸也没有浪费，都捐献给某某某了！”

像这样的例子还有很多。这些例子都表明，微软的员工已经将自己看成是公司的主人，如果公司有浪费行为，就应该主动提出来。每个员工都意识到，个人的成败与公司的兴衰是紧密联系在一起。

我认为，微软的主人翁精神包括了以下几个方面的内容。

1. 首创精神 (Initiative)

首创精神意味着要时刻准备好抓住新的机会。机会总是存在的，但是只有聪明人和有准备的人能及时地抓住它们。在抓住一个机会以后，你要确定一个相关的问题，得到一个很好的想法，这个想法对整个公司或者自己的团队都会有帮助。然后你就可以将这个想法“卖”给别人，以求得到经理、同事甚至公司的支持。对于一些人来说，这一步是最困难的，我们会研究，能得到许多好的想法，但不会“卖”想法。

下面我举几个例子进行说明。

案例一

第一个例子讲的是比尔·盖茨的前任个人助理 Russ Seligman。有一天他找到比尔·盖茨，说：“比尔，现在 AOL 发展太迅速了！我觉得微软需要制定一个策略，在互联网方面的业务上谋求发展，从而与 AOL 进行竞争！”比尔·盖茨听了以后，觉得非常有道理。在经过深思熟虑之后，比尔·盖茨接受了 Russ Seligman 的想法，于是 MSN 就这样诞生了。

注：AOL 指美国在线公司，MSN 指微软网络在线服务。

案例二

第二个例子是我自己的亲身经历。那是在 Visual Studio 开发团队的时候，我们的产品中需要增加一种 Graphic Layout 功能，即能够将数据库中的数据以图形的方式显示出来。当时微软还没有这方面的成熟技术，于是 Visual Studio 的产品经理就同一家公司进行谈判，准备获得这家公司的授权直接使用他们的产品。这是因为微软有这样一种策略：如果需要具备某种新功能的产品，那么能买到就买；如果买不到再考虑重用以前的代码；如果实在无法重用以前的代码，才会考虑编写代码来实现。

这项交易数额达 100 万美元，而且微软只能被授权使用该产品的执行代码，而不能获得源代码。当我们的产品经理同这家公司进行谈判的时候我并不知道。后来我一听说这项交易，就马上找到产品经理，要求看一下那家公司的产品具有哪些功能。看完以后，我对产品经理说：“不要买这家公司的产品，我们自己也能做出具有同样功能的产品。”



但当时微软没有这样的产品，又能买到，所以产品经理不太同意我的想法。于是我让他给我三天的时间考虑一下。在这三天里，我冥思苦想，反复修改设计方案，最终拿出了一份初步方案。我给产品经理看了以后，他觉得非常有道理，于是又给了我几天时间对这个方案进行完善，同时把与那家公司的合同签订时间推迟了。我一步一步地对方案进行了充实完善，同时对产品经理说：“这家公司没有提供该产品的源代码，以后出现问题或者升级时都要受制于人。如果该公司倒闭了，那我们就更麻烦了，很有可能会导致我们不得不重新开发该产品。因此，还不如我们现在就自己开发出来，况且，实现起来并不是很困难。”于是产品经理就找了七八个高级经理一起来研究我的方案，最后他们都觉得我说的有道理，于是当场就决定取消同那家公司的交易，由我们自己来开发。

由此可以看出，如果有一个机会摆在那里，你看到了那个机会，觉得自己能够做出一点贡献，就应该勇敢地站起来，说：“嘿！我可以做这件事！（Hey! I can do it!）”或者说，如果你有一个非常好的想法，一定要敢于向你的老板、同事说出来，使得他们接受并支持你的想法，然后你再实现这个想法，这样你不仅会取得非常大的成功，而且还会赢得宝贵的信用。

2. 领导能力（Leadership）

Ownership 也是一种 Leadership。

◀ 你首要的任务是要使产品获得成功。

我经过观察，发现一些中国学生缺乏这样的一种精神和意识。通常是领导或老板推一下，他们才动一下，不推就不动；或者他们



根本就不知道该怎么去做。实际上，当你加入了一个产品的开发团队以后，无论团队是只有一两个人，还是成百上千人，你都是该产品的一个负责人（Owner），都要有这样的一种精神和意识：“这是我的作品！我将制作它！我希望它能成功！（This is my baby! I will born him! I want him success!）”然后经过周密的计划、安排、研究、协作来完成它。

◀ 要积极地与自己的合作伙伴协作。

尤其要注意的是，在开发产品时，一定要利用所有可以利用的资源，包括你所在团队以外的资源。微软的软件资源是非常丰富的，因此，我们在开发产品的某项功能之前，一定要找到自己的合作伙伴，看他们那里是否有现成的技术，如果有，就直接拿过来用。

◀ 做一个果断的问题解决者。

我在研究院的时候，发现有些人比较喜欢埋怨，总是说：“这个东西我没有做完，是因为某某的原因……”其实埋怨对解决问题根本是一点用也没有的，反而会造成许多负面影响。在开发一项技术时，如果你所依赖的一个合作伙伴没有做好时，你不应该去抱怨，而应该主动去帮助他，因为帮助他实际上也是在帮助你自己。在美国的文化当中，喜欢埋怨是一种非常坏的表现。美国人喜欢问题解决者，他们希望在给你一个问题的后，无论你遇到什么样的困难，你都能够很快很好地解决。另外，当你在一个产品的开发中是一个组长或经理，领导了一些员工的时候，一定要记住：把你所领导的员工的成功看成是自己的成功，而绝对不要把他们的成功看成是对自己的挑战。在这一点上，美国的文化和中国的文化也有一些区别。

◀ 要勇于做自己技术的代言人。

我觉得这一点许多中国人做得都不好，包括我自己。我记得在1992年的时候，我在比尔·盖茨的私人公司里开发出了一个非常好的技术，直到现在我还觉得非常自豪。当时公司的另外三个人花了三个月来研究这项技术，也没有取得成功，而我只用两个星期就研究出来了。后来微软的技术人员过来参观我开发的这项技术的时候，

我却没有勇气站上去给他们讲，最后是我的老板给他们演示讲解的。当时我在下面听老板讲的时候感觉非常不好，觉得是自己开发出来的，为什么要让他来讲？后来等我也做了经理（Manager）的时候，我就体会到，其实当老板的也不愿意替自己的员工去讲解。当你开发了一项技术时，你一定要知道如何去为它代言，勇于告诉别人这项技术是我做的，甚至在有一个不相关的人要为你的技术代言时，你都要勇敢地站出来说：“不！那是我的！（No! That's mine!）”，而不要觉得害羞。

3. 勇于做决定（Decision）

这是中国学生普遍存在的一个很大的问题。他们不愿意做决定，不愿意说“Yes”或“No”；或者无论遇到什么意见都说“Yes”。我们在成立研究院半年之后就发现了这个问题，并领略到了这个问题的严重性。因为大家很难沟通，不知道每个人都有什么样的想法。于是我们赶紧召集大家开了一个会，给大家讲微软的企业文化，强调一定要学会说“No”，即使对自己的老板，甚至老板的老板。这一点对于美国文化根本就不是问题，我在美国这么多年，从来就没有遇到过因为某一个人对他的老板说了“No”而遭到什么不幸。下面我举两个我亲身经历的例子。

案例

2000年春天我在研究院的时候，当时要做几个演示（Demo）给比尔·盖茨看。我做完以后就把这几个例子发E-mail给微软亚洲研究院前任院长、微软现任副总裁李开复博士，请他看一下。开复看完以后给我回了一个很长的E-mail，大意是说其他的Demo都很好，但有一个Demo存

在问题，并列出了五条理由，要我根据他的意见进行修改。我看了以后就马上给开复回了一个 E-mail，我认为其中四个理由都是涉及增强软件的可观性和丰富程序内容方面的，而有一个理由却是关系到程序的稳定性问题，与软件是否能正确运行密切相关的。在产品最后发布阶段，时间的紧迫性是必须考虑的，因此，我对修改程序稳定性的理由说了“Yes”，而对其他理由说了“No”。

为什么呢？因为我拥有这个项目，我的 Ownership 决定了我对这个项目负有全权的责任，我懂的比别人多。我的任务就是保证按时完成项目，如果有人干扰我按时完成项目，即使这个干扰来自我的老板，我也要站起来说“No”。

当然，并不是说我们总是要对别人甚至自己的老板说“No”。当你的老板、同事甚至下级对你的工作提出了意见，你要仔细思考和分析。如果他们说的是对的，你也应该很愉快地说“Yes”。

4. 负责和牺牲精神 (Responsibility and Sacrifice)

在微软，当产品快要发布的时候（通常是在 Beta 2 阶段），每个工程师都会随时待命。不论白天黑夜，在任何时候，只要公司打来一个电话，你都要立即赶到公司来解决。这在微软被称做“发布前随时待命 (On-Call Before Shipping)”。微软人都知道，因为你拥有一个产品，你就必须对它负责。当然，这可能会需要牺牲自己的时间和个人爱好。我在微软 8 年的时间当中，只被 On-Call 过一次。那是因为 Visual Studio 产品中出现了一个 Priority 1 的 Bug。微软把 Bug 按照其危害程度分成 Priority 1, Priority 2 和 Priority 3 等类型，其中 Priority 1 是最严重的 Bug，即会导致程序崩溃的 Bug。遇到这样的 Bug，微软的要求是不修复好就不能回家。那一次我在办公室连续工

作了 24 个小时，不睡觉，不吃饭，直到最后彻底解决问题。

5. 回报 (Reward)

当然，微软的 Ownership 也是有回报的。

首先一个回报就是信用 (Credit)。在美国，人们非常在乎信用，甚至视它为生命。美国人没有档案，但每个人都有一个信用记录，这个信用记录是全国联网的。在微软，如果你能够成功地发布一个产品，那将是一个非常了不起的事情，非常令人自豪的事情。一旦项目成功，你就会得到宝贵的“信用”，所有人都知道这个成功属于你，你的能力得到了体现，你的辛苦得到了回报。你一次又一次积累起你的信用，你就会慢慢成为这个领域的旗帜，或者称做“精神领袖”。所以，你的信用不是由你的领导树立的，也不是舆论吹出来的，而是靠自己建立的。这同一般意义的“主人翁”不一样。你付出很多，得到也很多。可惜我们国内的很多研究人员特别缺乏这种概念。他们拥有一个项目之后，不知道这意味着什么，不知道应当怎样对待它，不知道他们做好这件事情后能够得到什么。其实，你得到的最大利益是信用，是你的满足感和自豪感，而不仅仅是金钱或者是往上爬的机会。

另一个回报就是前面提到的 Stock Option (配股权)。大家都有 Ownership 精神，公司自然就会发展得更好，业绩更理想，股票市值就会上升，这样每一个员工都会从中受益。

当然，对许多人、包括我自己来说，最大的回报可能就是自我满足的心理。看到有成千上万的人在使用自己开发出来的产品，自己的工作影响了成千上万人的生活，那种自豪的心情和成就感是无可名状的。对我来说，最欣喜的事莫过于在某个项目有所眉目时自豪地对别人说：“That is my baby! (那是我的成果!)”



2.5.2 团队精神

Ownership 绝对不是个人英雄主义或独裁，其中还包含了一种团队精神。在当今这个信息社会，仅靠单打独斗是行不通的。尤其是在现代的软件生产当中，团队精神是非常重要的。Ownership 并不意味着，如果你拥有一项技术，则一切都是你个人的。可以说，在现代软件的构造中，你做任何事情都要和其他人合作。你必须学会和别人合作，懂得怎样充分整合你能利用的一切资源，懂得和别人协调共事，懂得和人家交流并互相帮助。

案例分析

我面试过这样一名博士生，毕业于国内最好的名牌大学之一。他非常聪明，回答问题非常快，思路非常宽阔，也非常具有创新能力。但是，谈到最后，他就向我指责起他的导师来。他的导师是国内最著名的计算机学者之一。他甚至跟我说：“我这辈子犯的最大错误就是找了这样一位导师。他不仅没有时间，也根本不花时间来指导我，完全靠我自己来研究。”我听到这番话，心里马上就做了决定，并向其他尚未面试他的专家发了一封 E-mail，告诉他们“此人不能用”。

原因在什么地方呢？我并不在意是这位博士生对，还是他的导师对，我更看重的是：你在自己的学习、工作中，能不能跟和你意见不一致的人一起工作，能不能保持一种宽容的心态。因为人与人的思维方式是不一样的，你在工作当中肯定会遇到和你意见不一致的人，这一点是毫无疑问的。关键在于，当你和他人意见不一致时，能不能和他们坐在一起，心平气和地互相讨论。如果你只是坐在那里抱怨别人，那是一点用也没有的。



在微软，几乎所有的经理和组长都懂技术，所以他们都懂得如何根据产品的特点和需要，将一个大的开发团队分成许多小的团队，并有机地将它们组合在一起。因此，尽管微软有许多非常庞大的产品开发团队，如 Windows 2000 有 5000 万行代码，3000 多个工程师，几百个小团队，但仍然能够有效地工作，并生产出优秀的产品来。

微软的团队绝对不是：

- ◀ 领导一挥手，大家跟着跑的团队。
- ◀ 能承受高度压力，但缺乏灵感和对企业缺乏信心的团队。

微软团队成功的秘诀有以下三点。

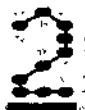
1. 微软团队的最大财富是智慧

团队的智慧是极其宝贵的，要比团队中任何个人的智慧大得多，但团队决不排除个人的智慧。当你要解决一个问题的时候，你要知道你的团队当中谁会干，谁最精通，谁对这个问题最清楚，而不是每次都老老实实地从第一步走起，这些身边的“财富”非常重要，懂得利用集体的智慧解决问题，从中学到更多的智慧。

我们中国人在几千年前就总结出了这种规律：“三个臭皮匠，赛过诸葛亮。”这比美国人要早得多。但在现实生活中，我们做得却比美国人要差得多。有人这样开玩笑来形容中国人：“三个诸葛亮顶不过一个臭皮匠。”虽然这样说有些夸张，但也表达了一些实际情况。中国人都太聪明，太聪明就不愿意合作，认为：“我不比你差，为什么要听你的？”，“听你的太掉份”等，甚至不光不愿意合作反而会互相拆台，最后两败俱伤，谁也无法成功。

微软的团队善于利用集体的智慧来解决难题。当一个员工遇到一个自己无法解决的问题时，他会将问题描述清楚，然后通过 E-mail 或会议等形式告诉大家，请大家一起来解决。通常很快就会有有人告诉他解决办法。反之，如果只是闭门造车，老死不相往来，则很有可能无法解决这个问题，或者解决问题花的时间太长，效率太低，





或者解决问题的方法非常不好。要善于了解并利用你身边的智慧财富。如果你自己去做每一件事，结果必定是既慢又不是最好的。

另外，还要承认别人的贡献，以便能接受别人更多的智慧。如果坚持强调“我的是我的，你的还是我的”，其结果必然是没有人愿意再帮你。

2. 微软团队的黏合剂是沟通

团队内部成员的沟通犹如团队良好运转的“润滑油”。

要积极地投入，为团队贡献智慧。如果本着“事不关己，高高挂起”的原则，结果必然会使团队变得死气沉沉，没有灵感，没有灵魂。要思想开放，从正面直接地提出建设性的意见，坦诚地、直截了当地想什么就说什么。如果瞻前顾后，生怕说错话，结果只会是变成谨小慎微的君子，而不是敢打敢冲的战士。要提供正面的、建设性的意见。如果这样评论一个意见：“真是一个垃圾想法，完全没用。”这对团队是无益的。要敢于说“No”，但要尊重不同的想法。如果不同意也不说出来，或者做极为负面的反对，结果只会是关闭了沟通的渠道。要学会倾听，抓住获得智慧的机会，而不能自说自话，不听别人说，或者过于注重表现自己。

微软团队的沟通最大程度地利用了 E-mail。我们很少跑到别人的办公室里去，一般都是通过 E-mail 来联系。

小故事

在微软有这样一个真实的故事：有一天下班的时间，一位女同事不小心把比尔·盖茨的凌志 400 给撞坏了，回去以后她心里非常不安，毕竟是把大老板的车给撞坏了。但她又不可能直接去找比尔，因为要和



比尔见面的话要经过很长时间的预约，于是她便给比尔发了一个 E-mail，说：“不小心把你的车撞坏了，我非常抱歉，请你能够原谅我！”一个小时以后，比尔就给她回了信：“没关系！你也不要担心了，至少你没有受伤，忘记这件事吧。祝你好运！”

3. 微软团队协作作战的保证是承诺

承诺的重要性体现在生活的方方面面，在工作中，你工作的成功是你对队友的承诺，队友的成功建立在你的承诺的基础上。成功团队最明显的特征就是互相帮助。要做任何可做的小事情去帮助团队成功。另外，互相信任也是非常重要的，它可以帮助你提高工作效率。相信你的队友做的承诺，要全心全意地实现你对队友做的承诺。

好的团队精神是一个好的软件开发的必然要求。好的团队，加上团队内成员高度的主人翁精神，将为软件开发的顺利开展铺平道路。团队的成功是个人成功的前提。如果团队失败了，更无从谈个人的成功。

2.6 锲而不舍、从错误中学习的精神

相信很多中国学生都有着永不服输的韧劲。锲而不舍是研究治学、软件开发等工作中宝贵的品德，但锲而不舍不是没有方法、毫无目的地死钻牛角尖。软件开发过程中，一个成功的软件产品的诞生，除了敏锐的洞察力和精湛的技术外，更需要开发过程中不断地





探索和永不服输的精神支柱。在微软，员工的这种锲而不舍、从错误中学习的精神是非常强烈的。大家都信奉任何优秀的产品都不是无法超越的，承认自己不够好，追求最好、更好是大家共同的信念。

小故事

有一次，微软的一位产品经理去参加一个颁奖大会。这次大会一共要颁 10 项奖，微软的这个产品就获得了其中的 9 项。这位产品经理回来后立即用 E-mail 把这个消息告诉了大家。24 小时之后，他收到了 40 封 E-mail，大家都是在问他：“我们的产品没有获得的那一个奖项是什么？”大家都认为：我们一定要做到最好，足够好是不够的。即使我们已经做到了最好，但仍然不够，还可以做得更好！

俗话说，“失败乃成功之母”。但是，真正做到敢于面对失败，承认错误，从错误中学习，是需要勇气和胆识的。我们不应该为自己犯的错误而抱憾不已，更不应为此而感到羞愧，甚至难于启齿。微软的成长也并非一帆风顺，对某些企业和人而言，失败带来的财富也不可估量。在微软，没有人会因为你的错误而疏远你，甚至怀疑你的本质能力，微软能够提供给你了解自己错误原因的机会和舞台，沟通拉近了彼此的距离，交流促进了你的改进。

在微软还有一个惯例：在一个产品已经发布（Shipped）以后，该产品的开发人员还要在一起开一个会，我们把这个会称做“Postmortem meeting”（检讨会）。会议的内容就是大家对这个产品进行反思，分析该产品还存在哪些问题，哪些方面做得不好，哪些方面还需要改进，然后把这些内容写成一个报告，送到上一级部门。

由此可见，微软的员工在做完一个产品之后，首先并不是急于



自卖自夸，而是急于查找产品中存在的问题。人家并不会因为自己的产品还存在不足而感到羞愧。

微软员工的这种锲而不舍、从错误中学习的理念所导致的结果就是：员工都愿意去尝试那些高风险、高回报的项目，而不会故步自封。相反，如果一个人犯了任何错误都会受到惩罚，那么他就会变得唯唯诺诺，不愿意去冒险，从而丧失创造性和主观能动性。

案例一

微软副总裁 Jon Devaan 曾经说过：“If you fire the person who failed, you are throwing away the value of the experience. (如果你解雇了一个曾经失败过的员工，你就是丢弃了一份经验。)”在他看来，失败 (Failure) 也是一种经验。事实上，这个观念现在已经深入到微软的企业文化当中了。微软绝对不会因为你做某件事情失败了而将你解雇。相反，微软会很珍惜你失败的这份经验。

案例二

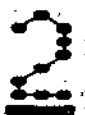
比尔·盖茨的技术顾问、微软高级副总裁 Craig Mundie，在加入微软之前，是一家软件公司的 CEO。但是那家公司运行得非常糟糕，最后宣布破产了。正是由于这个原因，比尔·盖茨将 Craig Mundie 招聘到了微软。因为比尔·盖茨认为 Craig Mundie 失败的经验对微软是非常重要的，正因为他失败过了，他才知道失败的滋味，并可以从中吸取教训。这样，微软借助他的经验，就可以避免犯同样的错误而导致失败。



总结

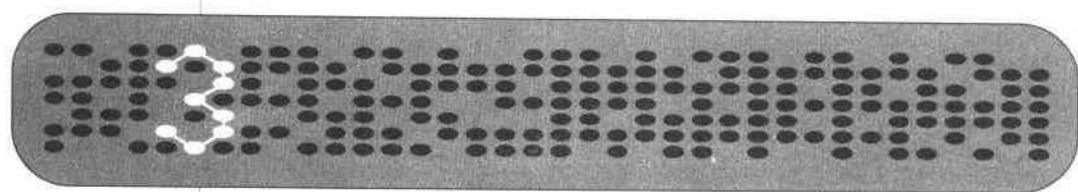
现代软件开发对我们的要求是方方面面的，微软在这些方面积累的经验 and 遵循的规则非常全面，在软件开发的整体过程中，从某种意义上其“艺术”性高于“科学”性，因为融入了“艺术”的产品有着超群的竞争力，而这一切又都归功于软件产品的主体——人。人的最大潜能的挖掘和团队最大可能的互助，必将创造成功的产品。

第 2 章



Chapter 2
Requirements for the New
Development
Modern software development
requirements for talent





第 3 章

从 研 究 到 产 品

From Research to Products

背 景

如何将研究成果投向市场并获得成功，这是许多人关心的问题。实际上，从研究成果到产品是一个非常复杂并且非常漫长的过程，其中会牵涉到相当多的问题。张益肇博士根据多年来积累的丰富实践经验，首先介绍了一种关于产品空间的思维方式——技术生命周期，以及在生命周期各阶段中用户对技术和市场的影响，其次通过具体的案例说明技术和市场的关系；最后总结出了三条宝贵的规则，相信会给读者带来很大的启发。

微软亚洲研究院主任研究员。主要研究方向是自然语言理解、机器学习和信号处理。目前在研究院主要从事自然语言理解方面的研究工作。

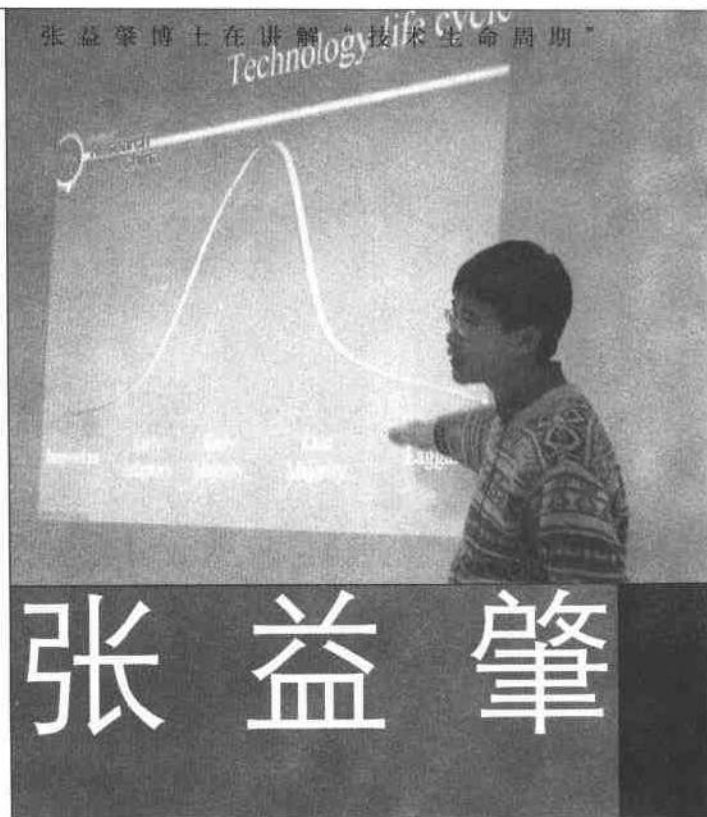
1990年 毕业于麻省理工学院 (MIT)，获电气工程和计算机科学学士及硕士学位。

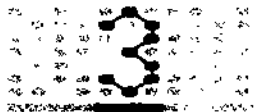
在麻省理工求学期间，他是著名的 Tau Beta Pi 和 Sigma Xi 荣誉协会的会员。

1995年 获博士学位。

张益肇博士曾先后在麻省理工学院林肯实验室、日本东芝 ULSEL 研究中心、通用电气研究中心从事产品和技术的研发工作：在 MIT，他开发出新的语音识别算法；在东芝，他发明了新的电路优化技术；在通用，他进行了模式识别方面的探索。此后他成为硅谷 Nuance Communications 公司研究部的创始人之一，正是基于张益肇与公司全体同仁的不懈努力，才使得该公司成为电话语音界面产定的先驱。

在张益肇加盟微软亚洲研究院之前，他已在国际著名刊物和学术会议上发表过多篇有关语音识别、神经网络和遗传算法方面的论文，同时还是多项专利技术的持有人。





本章内容概览

- ☞ 引言
- ☞ 技术生命周期
- ☞ 案例分析
- ☞ 练习
- ☞ 间断技术
- ☞ 基本规则
- ☞ 推荐书目

3.1 引言

3.1.1 三个问题

如何将研究成果投向市场并获得成功，这是许多人关心的问题。实际上，从研究到产品是一个非常复杂的过程，其中会涉及到相当多的问题。下面我根据自己积累的一些经验，讲一讲个人的一些想法。

把一个研究的成果变成一个真正可用的产品，有一段非常漫长的路要走。在这个转化过程中，最常见的三个基本问题是：

◀ 同样都是非常优秀的技术，为什么有些技术得到了广泛的应用，从而走向成功；而有些技术却不被人们接受，从而走向失败呢？有些技术具有非常好的应用前景，为什么它们却无法在市场上走向成功？

◀ 你已经发明了一种非常好的技术，怎样才能让它成功地变成





好的产品，最容易地走向市场呢？如今，越来越多的年轻人（尤其是刚刚毕业的学生）都走上了自主创业这条路。怎样才能将实验室的技术成果转化为具有竞争力的产品，成功地推向市场？

◀ 怎样来评估一项新出现的技术设想？无论是你希望成立一家公司，或是你目睹别人成立公司，又或是你即将加入一个新的公司，你都需要回答这个问题：如何才能正确地评估一项新的技术是否成熟，它在市场上的前景会是怎样？是否能马上推向市场，还是需要继续进行研究？

下面我就围绕着这三个问题来进行讲述。相信读完本章，无论是你自己想创业，还是你的一位朋友满怀激情地对你说：“我研究出了一种新的技术，我要马上去开公司！”，此时你都可以根据本章所学到的技巧，正确地判断自己或朋友的研究成果是否能够成功。

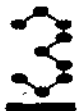
3.1.2 我的主要经历和体会

由于本章主要取材于我个人的实践经验，因此我首先来谈一下自己在“把研究转换成产品”这方面所做的研究、主要经历和体会。

我在麻省理工学院（MIT）拿到学士和硕士学位以后，在许多大大小小的不同公司工作过。

◀ 从人数和历史上来讲，最大的公司就是美国的通用电气研究中心。它成立于 19 世纪，是由爱迪生创建的实验室逐渐发展而来的研究机构。通用电气研究中心的大部分员工都是拿到了博士学位并进行了很多年研究的研究人员。我在工作中就发现，有许多人都能够很快地把研究成果转换成产品，并拿到市场上去应用。但是，同样也有一些人却找不到转换的窍门，无法把研究出来的非常深奥的成果转换成人们可以理解的产品。

◀ 后来我在日本东芝 ULSEL 研究中心从事用神经网络来设计电子回路系统的研究。这套系统目前还没有产品化。在这个过程中我也得到了一点体会：在某些情况下，将研究的成果转换成产品的过程非常困难。稍后我会具体地讲述这一点。



◀ 接下来我在麻省理工学院（MIT）林肯实验室做博士论文，主要研究任务是把一个人的特定语音从一长串的话中搜索出来。例如，如果要做一个新闻节目的搜索，需要从今天的广播中把所有包含了“奥运”这两个字的发音的新闻语句都找出来，以便能够很容易地听到自己感兴趣的新闻信息。这时就可以采用我所研究的技术。这项研究的应用范围比较有限，主要应用于军方的情报侦察中。尽管如此，从这项技术所转化出来的产品仍然很实用。

◀ 然后我加入了硅谷的 Nuance Communications 公司，它是目前美国最大的一家从事电话语音系统业务的公司。通过这种电话语音系统，人们可以很方便地享受股票行情查询、订购电影票等多种服务。在此期间我参与了多个项目的研究工作，包括自信度生成、声学模型和顽健型语音检测等等，并领导公司研究人员开发出世界上第一个自然日语语音识别系统——Nuance 的日文版本。我加入的时候是公司的第 15 位员工，到 1999 年我离开的时候已经有 300 多名员工了，其每年的营业额也从最开始的 100 万美元上升到现在的 5000 万美元。从这个例子也可以看出：如果能够迅速地把一项研究成果转换为产品的话，那么公司的发展速度将非常快。

◀ 现在我加盟了微软亚洲研究院，目前我们所做的一些研究已经接近产品化了。在这个过程中，我也得到了许多的经验。（注：微软亚洲研究院所研究的中文语音技术，如今已转移到微软 Office XP 中。）



总结

我最大的心得是，一个产品一定要找到能够真正适用的场合。不能只是为了技术而从事技术，为了研究而进行研究，却不管用户对你所研究的技术和产品有没有需求。否则，无论你的技术是多么优秀、多么先进，恐怕你的产品在市场上都无法获得成功。



- 下面将讲述以下几个方面的内容：
- ◀ 一项技术研究出来以后，为什么有些会成功，有些会失败？
 - ◀ 一项技术从研究出来到最后被新技术所取代的过程如何？
 - ◀ 根据我多年的经验所得到的三个规则——一项新技术要成功地推向市场所应遵循的规则。

3.2 技术生命周期

为了解决以上三个问题，首先要介绍一种关于产品空间的思维方式——技术生命周期，以便解释为什么一部分技术转向市场时取得了成功，另外一部分的转化却失败了。

所谓技术生命周期，实际就是指把一项技术变成产品并推向市场所经历的过程。杰夫·摩尔（Geoff Moore）在他的著作 *Crossing the Chasm* 中引用了如图 3.1 所示的技术生命周期曲线图。

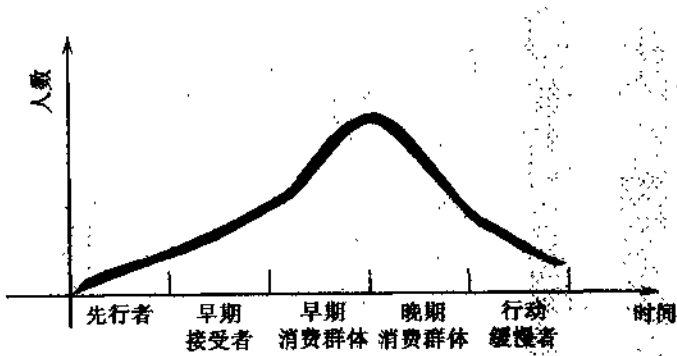


图 3.1 技术生命周期曲线

其中横坐标代表技术和产品的发展阶段，纵坐标代表各个阶段使用该产品的用户人数。这个曲线将技术生命周期中的用户按照发展阶段分成了五类：



◀ 先行者 (Innovators): 希望尝试任何新的技术和产品。

◀ 早期接受者 (Early Adaptors): 愿意尝试新的技术和产品, 但是一定是基于某些目的的尝试。

◀ 早期消费群体 (Early Majority): 等到新技术已经被人们广泛接受了以后, 他们才会介入。

◀ 晚期消费群体 (Late Majority): 接受一项新技术是因为不得不这样做, 因为这些产品已经成为一种必需品。

◀ 行动缓慢者 (Laggards): 非常顽固地不愿意尝试任何新的技术和产品。

从这个曲线的走向可以看出, 先行者和行动缓慢者的人数都是很少的, 大部分的用户都集中在中间的两个群体中。这是为什么呢? 首先我们要清楚上述五种不同的用户之间的区别及各自的特性。

3.2.1 先行者 (Innovators)

每当一种新技术出现的时候他们都会最先尝试, 即使这时候刚刚推出的新产品价格非常昂贵。他们是真正的“发烧友”, 对新技术抱有极大的热情。对他们而言, 技术就是纯粹的技术, 越是难使用的技术他们越有兴趣去尝试, 也越有成就感。这些人乐于在普通人之前享受新技术带来的乐趣。

案例一

例如一些音响发烧友, 他们总是不惜一切去寻觅最新最好的音响器材, 追求最完美的效果。有时候我们甚至不能用常理来看待他们, 如要使用几百美元一根的音频线, 并且不能用圆形的, 必须使用方形的等等, 以求得声音上的一点改

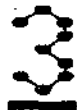
观。而当这样的器材渐渐普及的时候他们又总会将目光投向更新的产品。对他们而言，这个过程就是生命的乐趣。这些人永远走在普通消费者的前列。

先行者的特点是：他们永远不会成为主要的用户群。如果一个产品的主要用户群是先行者的话，那么这个产品的市场份额非常有限，因为这样的人很少。尽管这部分人并不多，也不会有太大的市场份额，但是这些人对新技术顺利走向下一个阶段的意义非常重大，他们会影响周围的人，推广这种新技术，从而带动那些会真正使用这项新技术的人。

3.2.2 早期接受者（Early Adopters）

早期接受者是一些比较有眼光的人。对他们而言，关注一项新技术并不是因为这项技术很新、很好，而是看这项技术对自己是否有真正的帮助。如是否能够给自己带来实际的利益；是否能够在生意上带来便捷，创造新的价值；或者是否能够帮助自己实现梦想，解决以前无法解决的问题等。早期接受者与先行者都不惧怕使用新的技术，但是他们之间最大的区别在于，早期接受者强调的是在新技术上的投入一定要能带来更大的产出。

有的厂家注意到了早期接受者的需求，因此他们就在这一阶段取得了成功。例如，世界上最大的电脑零售商戴尔（Dell）公司，是第一个在网上进行销售的厂家。他们提供了一个产品目录，消费者可以按照自己的需求选择不同的硬件进行组装，配置自己的个人电脑，如可以选择不同的 CPU、内存、硬盘、显卡等，以适应个人的需要。采取这一策略对戴尔来说也是一个风险，但是戴尔意识到这样做会使他们在电脑销售上领先对手一步，从而获取更大的利润。



果然，当人们逐渐习惯并喜欢上这种销售方式以后，戴尔的销售份额大大提升，打败了康柏（Compaq）等竞争对手而获得了成功。

又如，在线股票交易和银行转账，尽管现在已经普遍使用了，但是在 1994 年、1995 年的时候还只是刚刚出现，安全性并没有太大的保障，在线进行股票的买卖和银行的转账是要冒很大风险的，如可能会被黑客攻击，造成损失。但是对于一些人来说，网上交易更加便捷，可以降低成本，从而带来更大收益，因此他们仍然愿意尝试，以取得在自己行业上的领先地位。

在把一项研究成果成功地转化为产品的过程中，早期接受者是最重要、最关键的一群人。因此，在把一项技术推广到市场中时，如果找不到早期接受者——一群很迫切需要这项技术的用户，而大多数人只是对这项技术感兴趣，但并不急于使用它，那么这项技术就永远只能停留在上一阶段，从而无法取得成功。

3.2.3 早期消费群体（Early Majority）

早期消费群体是实用主义者。他们绝对不会第一个使用一项新技术，而是要看到同行业的其他人使用这项新技术已经取得了成功，或者等到这项新技术已经被人们广泛接受了以后，他们才会介入。换句话说，这群人是随着大部分人的行为而有所行动，大家怎么做，他们也怎么做，只要不落后就可以了。所以在这个阶段，产品的销售量将会有很大的增长。也正是因为这时技术已经非常成熟了，很安全，从而开始拥有相当广泛的使用者。

案例

例如，一开始只有很少的公司有他们自己的主页，但是后来主页被广泛接受了，大家都意识到：如果一个公司没有



自己的主页，似乎就不是一个正规的公司。于是，在不到一年的时间里，超过 50% 的公司都建立了自己的网站。另外一个很好的例子就是免费 E-mail，最初只有雅虎（Yahoo）等少数网站提供免费 E-mail 服务，但是目前免费 E-mail 服务已经到了“Early Majority”的阶段，大家都认为如果一个门户网站连免费 E-mail 服务都没有，肯定不是一个成熟的网站。因此，现在几乎所有的门户网站都提供自己的免费 E-mail 服务。可以说，这个时候已经到了“别人都在这样做，我不做不行”的阶段了。

早期消费群体是产品最大的一个市场。这时早期消费群体的数量已经至少 10 倍于早期接受者的数量了。把产品的用户从早期接受者转移到早期消费群体，需要一定技巧。同时，这也是一个产品能否在市场上取得成功的最关键所在。稍后将具体讲述跨越这两个阶段的技巧。

3.2.4 晚期消费群体（Late Majority）

晚期消费群体是一些非常保守的人。这些人从不试图去改变些什么，也很不愿意使用新的技术。他们认为，如果一个东西仍旧可以用，那为什么需要改变呢？因此，当一样东西还没有彻底损坏之前，他们绝对不会主动去用新的东西来取代它。所以说，他们是典型的保守者。但是，如果别人要求他们使用一项新技术，他们还是会接受。对他们而言，之所以接受一项新技术是因为不得不这样做，因为这些产品已经成为一种必需品。当等到市场中的早期消费群体已经饱和时，就表明整个市场中的一半用户已经接受这一新技术，这时使用该技术已经是大势所趋。要想融入主流社会，就必须使用





该技术。

案例一

例如，在 20 世纪 80 年代的时候，几乎所有的报告都是用手书写在纸上，但是同样也有一部分早期接受者在家中采用计算机或是打字机来写报告，这样当然更加方便和迅速。现在相当部分的文档是用 Word 文档递交的，人们很少使用完全手写的报告。特别在公司或政府机关这样的范围内，员工基本上不再被允许递交手写的报告。这就表明文字处理技术已经发展到了“Late Majority”（晚期消费者）阶段，即使是一些对这项技术不感兴趣的人，也不得不使用这样的方式进行工作。

案例二

传真机的普遍使用也经历了这样的一个过程。最初只有一些特定的机构（如报社）使用传真机，而到现在传真机几乎已经成为任何类型公司的必备办公用品。如果有哪家公司还没有使用传真机，大家一定会感到不可思议。

其他的例子还有很多，如电子邮件（E-mail）信箱、手机等的广泛使用。现在在美国，不用说公司中的白领，即使是家庭主妇也有自己的 E-mail 信箱。人们和外界联系经常采用 E-mail 信箱，它的重要性与电话已经不相上下。这说明 E-mail 信箱的使用在生活中已经是无处不在，已经进入到“Late Majority”的阶段。大家可以回顾一



下自己从“没有 E-mail 信箱”，到“受别人或其他因素影响开始使用 E-mail 信箱”，再到“推荐别人使用 E-mail 信箱”这样一个过程，从中也可以看出一项新技术的生命周期及发展过程：开始时是被大家试用，后来变得很成熟，更多的人开始使用，最后被大家广泛使用。

3.2.5 行动缓慢者 (Laggards)

这些人总是保持着一种怀疑态度，不管新技术多么好，他们都会拒绝接受，觉得自己不需要。例如，他们总是质问：“为什么一定要使用呼机、手机，让别人知道我在什么地方呢？为什么要用 E-mail 信箱呢？用信纸和钢笔来写信不是更具有亲切感、更能表达感情吗？为什么生活不能单纯一些呢？没有计算机等高科技产品我不是一样活得很好吗……”他们觉得没有新技术同样可以很好地生活，或者从另一个角度看，新技术对他们而言实在太难，无法掌握。他们宁愿生活在一个使用信纸和钢笔来写信的、没有计算机和通信设备的环境中，不但不去享受新技术带来的方便和愉快，反而认为新技术会给生活带来不利影响。显然，在现代社会中，这些行动缓慢者很难和别人进行良好的沟通。幸运的是，这样的行动缓慢者毕竟只是少数。

如果一项技术已经到达这一阶段，说明这项技术已经发展得差不多了，没有更大的潜力和发展空间，其产品的占有率已经很难增长了，因为剩余那部分行动迟缓者使用新技术的可能性非常小。此时，我们最好转而关注其他更有上扬空间的技术成果。

3.3 案例分析

上面具体阐述了新技术的生命周期的不同阶段，以及在这些阶段中用户对技术和市场的影响。下面结合一些具体的案例来说明一下技术和市场的关系。



3.3.1 手机

手机是摩托罗拉（Motorola）公司在 20 世纪 70 年代初期提出的产品，很明显，这个产品获得了相当大的成功。

1. 先行者

最早拥有手机的是那些非常富有的人。他们不会过多地考虑价格的因素，对他们而言拥有手机是身份的象征，比如手机在香港被称为“大哥大”就是这个原因。手机这个产品在刚刚推出的时候，价格比较昂贵，因此市场还不火。

2. 早期接受者

等到人们逐渐接受了手机这个产品，认为手机不仅能够表现身份，而且的确能给他们的生活带来方便的时候，手机的早期接受者就出现了。例如，其中一种用户就是那些销售人员。对他们而言，使用手机可以更加便捷地和客户联络，从而更快更多地销售自己的产品，获取更大的利益。由于销售人员要经常在外面跑业务，如果没有手机，客户就可能联系不到他们，这样就会失去很多的交易；而如果使用手机的话，销售人员在任何时间、任何地点都能和自己的客户取得联系。相比较而言，他们在购买和使用手机上所花的费用比没有手机所失去的交易额要少得多，因此他们使用手机主要是基于商业上更多利益的考虑，而并不是为了代表身份和高品味。

3. 早期消费群体

随着手机价格的下跌和性能的提高，更多的人因为公务加入到购买手机的行列。当他们离开办公室外出时，希望能够和单位保持联系，或者使用手机找到别人，而不仅仅是希望别人可以通过手机找到自己。这个时候手机的市场就开始变得很大了，目前中国手机市场就处于这个阶段。



4. 晚期消费群体

大多数消费者都会购买自己的手机。他们使用手机和自己圈子里的朋友联系，相互之间发短消息。在这一阶段手机完全是为了个人联络的方便，而不是出于商业的目的。目前美国、欧洲、日本等国家的手机市场就处于这个阶段。无论是十几岁的小孩子，还是八十多岁的老人，都在使用手机，这已经是司空见惯的事情了。据统计，在芬兰，手机的普及率已经达到了 80%，可以说除了无法使用手机的人（如 6 岁以下的儿童、残疾人等）之外，几乎每个人都拥有一部手机。到了这个阶段，表明手机这个技术已经完全融入了这个社会，人们已经普遍意识到了利用手机进行移动通信的方便之处。手机已经成了人们日常生活中必不可少的一部分。

5. 行动缓慢者

不管手机的普及率有多高，总是有一些人不愿意接受手机这一产品，他们或者不想被手机干扰生活，或者是觉得携带手机是一件很麻烦的事情。

3.3.2 个人电脑

1. 先行者

这些人是一些计算机的热衷者。他们在自己家中添置计算机，喜欢通过编程看到程序的各种不同的执行结果。这个阶段的用户使用计算机与商业目的无关。

2. 早期接受者

早期的接受者使用个人电脑来帮助自己完成各种复杂的工作，如进行电子表格的处理。在 20 世纪 70 年代末的时候，一些公司使用诸如 Visicalc 等电子表格进行一些简单的计算和数据统计。例如，



可以根据大量的数据计算统计出公司的销售额、年利润，以及二者之间的关系等；可以估算出如果销售额提高 10%，最终年利润可以提高多少……如果没有计算机的支持，这样的工作十分烦琐（需要浪费大量的人力、物力和时间）。尽管购买个人电脑及电子表格软件会花费一些钱，但是反过来使用电子表格可以非常方便地计算各种财务报表，建立公司未来的模型和计划，估算成本和利润的关系，从而可以带来更大的收益。正是基于这样的一些原因，使得个人电脑广泛地被各种专业人士所使用。

3. 早期消费群体

传统的计算工作是由会计部门负责的，这些工作人员都愿意使用计算机来管理数据。后来随着计算机技术的发展和性价比的提高，很多的白领也开始在他们的工作中使用电脑，进行简单的数据库操作、文字处理等。例如，很多公司都用数据库管理所有工作人员的记录，这样比用文件管理方便有效而且快捷，特别是具有检索、分类、统计和打印等功能。这样，市场就慢慢地越变越大。

4. 晚期消费群体

大约在 1995 年，个人电脑开始进入家庭。在美国，有超过 20% 的家里有不止一台电脑。家庭的不同成员使用个人电脑的用途也各不相同。父亲可能要将没有做完的工作拿回家来做，通过 E-mail 和更多的人联系；母亲可能要通过个人电脑上互联网查询一些信息；小儿子可能要用电脑玩游戏。这时电脑已经成为家庭的必备品。

5. 行动缓慢者

在美国大约有 10% 的人不使用个人电脑。他们或者认为电脑没有用处，或者觉得学习电脑太难，或者认为支付购买电脑的费用划不来。对于这些人，产品的开发人员无须投入过多的关注。





3.3.3 数码相机

1. 先行者

早在 20 世纪 80 年代初期，索尼（SONY）公司就推出了它的第一台数码相机，那时由于价格非常昂贵，而且图像的清晰度也不高，只有少部分的先行消费者购买。如同 1999 年索尼公司推出的网上销售的机器狗玩具一样，尽管每个售价在 2 000 到 3 000 美元之间，但是仍旧有一些对新鲜事物非常感兴趣的先行者购买。

2. 早期接受者

后来随着技术的发展，数码相机的清晰度得到了很大的提高，尽管这时的数码相机仍旧十分昂贵，但是仍有不少早期接受者使用它们，这些使用者都是急需这种功能的消费者。比如，记者就是数码相机的早期接受者之一。使用数码相机，记者可以更加及时地将拍摄的照片传送回报社，抢得宝贵时间。例如，一位采访奥运会的记者，要在第一时间将比赛的照片传送回去登稿。如果使用传统相机拍照的话，需要花费很长的时间进行洗相和邮递，到最后“新闻”往往都变成“旧闻”了。这时候如果使用数码相机就可以帮很大的忙，通过数码相机采集的照片可以通过网络或是电话线快速地传送回报社，真正做到新闻的及时性。对记者们而言，时间才是最重要的，价格反而没有那么重要了。

3. 早期消费群体

随着价格的下跌和性能的提高，数码相机逐渐变得普及起来，许多拥有个人电脑的人成了数码相机的早期消费群体。例如，使用数码相机可以很方便地将照片下载到计算机，然后通过网络发送给自己的亲朋好友，或者上传到自己的个人主页上，让大家都可以非常方便地共享照片。甚至还可以使用计算机对照片进行各种艺术处



理，并打印出来。目前，数码相机在美国正处于这样的阶段，很多个人电脑拥有者都开始购买数码相机。对一些摄影商店来说，数码相机也很有用。摄像人员可以随时删除不好的照片，只保留好的照片。

4. 晚期消费群体

这一阶段目前还没有到来。但大家可以展望：当数码相机的成本进一步降低后，会有更多的人拥有数码相机。因为这时数码相机的成本可能比普通相机和底片的成本还要低些，所以会得到更多的用户。我相信在两年之后的美国，买传统相机的人肯定会少于买数码相机的人。在这个阶段，数码相机已经非常普及了，逐渐成为人们生活的必需品。

5. 行动缓慢者

这部分消费者拒绝接受数码相机，排斥它所带来的任何改变。

3.4 练习

为了进一步加深大家对上述问题的理解，下面我将给出一些练习，让大家现学现用。请大家分别针对前面列出的五种用户来仔细分析这些技术和市场的关系，以便掌握正确评估一项新技术的方法。

3.4.1 题目

我给出的练习题目有下面三个，不过大家也可以选择其他的题目来进行练习。



1. 可视电话

早在 20 世纪 50 年代，美国的贝尔（Bell）实验室就已经开始展示能够看到通话双方图像的可视电话产品了，到了 1990 年美国的 AT&T 公司推出的可视电话已经非常成熟，而自 1995 年微软推出 Netmeeting（网络会议）软件以后，人们已经可以借助摄像头通过互联网来打可视电话了，如今第三代的手机中也已经加入了可视电话的功能。尽管可视电话的技术发展得越来越好，但是普及率却一直都不高，所以请大家分析一下这是为什么？这项技术的未来发展情况如何？怎样才能保证可视电话将来会得到广泛的应用？

2. 虚拟现实

虚拟现实技术已经研究了 20 多年，但目前还不是很成熟。我想让大家分析一下，在虚拟现实技术还不是很完善的时候，怎样才能让该技术能够更广泛地被人们应用？

3. 地理信息系统

地理信息系统（GIS, Geographic Information System）是通过综合使用数字化图像和高级数据库技术，使用户摆脱纸质“文字”、“地图”限制而存储、应用信息的系统。它以计算机为手段，对具有地理特征的空间数据进行处理，能以一个空间信息为主线，将其他各种与其有关的空间位置信息结合起来。它的诞生改变了传统的数值处理信息方式，使信息处理由数值领域步入空间领域。地理信息系统的用途十分广泛，可以为各类应用目的服务，例如交通、能源、农林、水利、测绘、地矿、环境、航空、国土资源综合利用等。作为“数字地球”的骨架支撑技术之一，地理信息系统关系到国民经济建设、社会发展和国家安全。请分析一下地理信息系统的技术与市场的关系。



3.4.2 习题解析

1. 可视电话

(1) 先行者

和手机类似，第一个阶段使用可视电话的应该是那些比较富有的人或玩家。他们在率先使用可视电话时主要追求的是一种使用新技术的快感，或者把使用可视电话作为一种身份的象征。简单地说，这些先行者追求的是使用可视电话给自己带来的一种快感和荣耀。这样的人不会很多。

(2) 早期接受者

在先行者的促进下，同时也随着可视电话的产品性能的提高，有一些人意识到使用可视电话能够给自己带来利益。因此，尽管可视电话还是很贵，但这些人为了谋求更高的利益，仍然愿意去使用。例如，一些大公司可以使用可视电话来进行远程会议、远程谈判；一些大的培训机构可以使用可视电话来进行远程教育；军方可以使用可视电话来进行实时性的探测和战略性的调度指挥。这个时候的用户主要是一些企业或部门，个人还无法承担可视电话的高额费用。

(3) 早期消费群体

随着价格的降低，一些中产阶级的个人和家庭也开始使用可视电话，这样，即使离家比较远，在平时或过年过节的时候也可以通过可视电话问候，再也不用遗憾因为没有时间回家而无法见到亲人了。但是，这个阶段的用户依然比较少。由于可视电话要实时传输图像信息，对电话线路和网络带宽要求很高，因此，尽管这个阶段许多用户能够承担得起可视电话的费用，但由于受到外界环境的影响仍然无法使用可视电话。



(4) 晚期消费群体

随着整体环境的改善（包括电话线路等外界条件的改善、可视电话技术的完善以及价格的降低），可视电话逐渐开始普及，并逐渐淘汰普通电话。

(5) 行动缓慢者

在这个阶段，可视电话的用户已经基本稳定，除了有一些人不愿意接受电视电话之外，我觉得更主要的原因是出现了比可视电话更好的产品。替代产品的出现是一项技术逐渐被淘汰的最主要的因素。

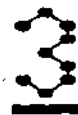
2. 虚拟现实

(1) 先行者

最早使用虚拟现实技术的人都是一些非常富有的人，他们非常乐于尝试虚拟现实技术带给个人的感官尤其是视觉上的新鲜感和刺激感。

(2) 早期接受者

此时虚拟现实技术的成本仍然很高，但是它在实际应用中能够带来许多好处。在一些非常危险或者成本很高，甚至当时无法实现的实验中，如果使用虚拟现实技术就可以很轻易地达到实验的目的，其好处是显而易见的。例如，军队、机场、宇航局等部门在培养飞行员或宇航员的时候就会率先使用虚拟现实技术，以获得很大的收益；影视制作部门也会使用虚拟现实技术来做一些特技，以达到更加震撼人心的视觉效果。但是在这个阶段，一般的部门还支付不起虚拟现实技术的高额费用。



（3）早期消费群体

等到虚拟现实技术的费用进一步下降以后，就会有更多的人开始使用它。例如，医院里的外科医生可以借助虚拟现实技术学习和提高手术水平。房地产商可以借助虚拟现实技术向人们更好地展示期房的规划、结构和布局。

（4）晚期消费群体

这时虚拟现实技术已经发展得非常完善，价格也趋于合理，越来越多的人都开始使用它，虚拟现实技术基本上普及到各行各业。例如，一般的商家可以将虚拟现实技术应用到自己的电子商务平台上，使得世界各地的用户都可以通过互联网真实地感受商家的产品；旅游公司可以使用虚拟现实技术为各个旅游景点作最逼真的广告，让人们有一种身临其境的感觉；技术设计领域的设计师可以借助虚拟现实技术来实时观察各种设计效果；另外，远程教学、远程医疗、三维游戏等领域也会广泛用到虚拟现实技术。

（5）行动缓慢者

尽管虚拟现实技术已经非常普及了，但仍然有一些人认为虚拟现实技术不实用，因而不愿意接受并使用它。

3. 地理信息系统

（1）先行者

地理信息系统起源于北美。世界第一个运行性地理信息系统是在 1963 年加拿大土地调查局为了处理大量的土地调查资料，由测量学家托明森（R.F.Tominson）提出并建立的。同一时期美国哈佛大学的计算机图形与空间分析实验室，建立了 SYMAP 系统软件，竭力发展空间分析模型和制图软件。这个阶段只有很少一些研究机构或人员对地理信息系统感兴趣。而且，由于当时计算机技术水平不高，



存储量小，磁带存取速度较慢，使得 GIS 带有更多的机助制图色彩，地学分析功能极简单。此时，地理信息系统还不实用。

(2) 早期接受者

20 世纪 70 年代以后，由于计算机软硬件迅速发展，特别是大容量存储功能磁盘的使用，为地理空间数据的录入、存储、检索、输出提供了强有力的手段，使 GIS 朝实用方向迅速发展。美国、加拿大、英国、西德、瑞典、日本等发达国家先后建立了许多不同专题、不同规模、不同类型的各具特色的地理信息系统。如美国地质调查局建立了 50 多个地理信息系统，用做地理、地质、地形和水资源等领域空间信息的工具。这期间许多大学和研究机构也开始重视 GIS 软件设计及应用的研究。这一阶段的 GIS 分析功能和 20 世纪 60 年代相比，并未得到很大的扩充，许多数据库的容量也较小。因此，20 世纪 70 年代可以说是地理信息系统的巩固阶段。

(3) 早期消费群体

20 世纪 80 年代是 GIS 普及和推广应用的大发展阶段，由于新一代高性能的计算机的普及和迅速发展，GIS 也逐步走向成熟。GIS 的软硬件投资大大降低而能力明显提高，已进入多学科领域，由功能单一、比较简单的分散系统发展成为多功能的用户共享的综合性信息系统，并向智能化发展。随着 GIS 与卫星遥感技术的结合，GIS 已用于全球变化的研究与监测，如全球沙漠化、厄尔尼诺现象等研究。所以，这一阶段的地理信息系统取得了突破性的发展。

(4) 晚期消费群体

20 世纪 90 年代，GIS 已成为确定性产业并渗透到各行各业，如交通、能源、农林、水利、测绘、地矿、环境、航空、国土资源综合利用等。投入使用的 GIS 系统数量每 2 到 3 年就翻一番。愈来愈多的国际性会议、学术刊物以 GIS 为主题，它已成为人们规划管理中不可缺少的应用工具。



（5）行动缓慢者

这时地理信息系统已经不复存在了，已经被功能更齐全、覆盖范围更广泛的“数字地球”系统所取代了。

3.5 间断技术

接下来要阐述如何把一项新技术成功地应用到实际生产和市场中去。值得注意的是，这里我所讲的技术指的是“间断技术”（Discontinuous Technology）。

间断技术是指用于替代旧技术的技术，它并不是对现有技术的简单扩展和再包装。

举例来看：英特尔公司每年都会推出速度更快、性能更好的 CPU，但是新 CPU 的基本框架并没有改变；另外，存储器芯片的容量也越来越大，芯片的存储能力每 18 个月会翻一番，但是存储的机制并没有改变。这些技术都是连续发展的，虽然也是新技术，但不是间断技术。而我要介绍的间断技术是指那些真正取代旧技术的技术，如汽车取代马车、电话取代电报等这些具有质的飞跃的技术。

3.5.1 鸿沟

其实前面介绍的技术生命周期曲线（如图 3.1 所示）只是一种理想的情况。事实上，一项技术从研究到产品的发展过程往往不会这么顺利，真正的技术生命周期曲线如图 3.2 所示。

从图中可以看到，技术生命周期曲线在早期接受者和早期消费群体之间存在一个鸿沟。许多技术由于无法跨越这个鸿沟，才导致在市场上的失败。也就是说，一项技术只有成功地跨越了这一鸿沟，才能成为一项被广泛应用的技术。所以对那些希望将技术转化为产品的人而言，如何跨越这个难以逾越的鸿沟，怎样跨入成熟期，就





是相当关键的一步。

那么为什么存在这个鸿沟呢？原因在于，除了先行者和早期接受者使用了这一技术以外，其他的消费者没有跟上来。当这个新技术出现时，首先先行者会进行尝试，然后早期接受者也会把它用到对自己有利的领域中。过了一段时间以后，这项新技术对先行者和早期接受者而言就不再新鲜了，于是他们就不再把兴趣放在这一技术上，转而去尝试和使用其他的新技术了。如果此时该项新技术对于早期消费群体（即大多数消费者）来讲仍然太复杂，即使早期消费群体看到早期接受者使用这项技术获得了利益，他们还是搞不清楚这项技术为什么好用，对他们到底有什么帮助，反而会觉得这项技术对他们来说不实用，因而也就不愿意接受并使用它。于是，鸿沟就出现了。

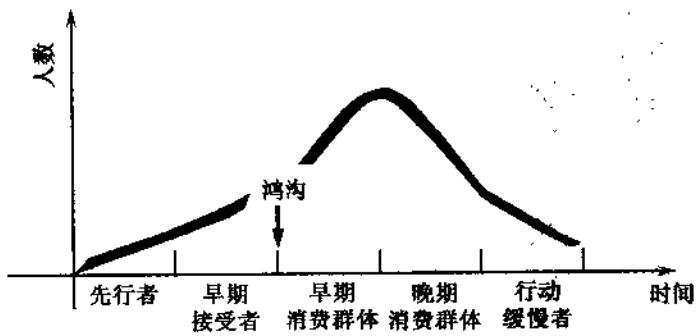


图 3.2 真正的技术生命周期曲线

案例分析

当几年前 World Wide Web 技术刚出现的时候，还只有基于文字界面的浏览 Web 页面的浏览器，这时只有一些对电脑





非常熟悉的用户能够通过这个文字界面在各种 Web 站点上浏览信息、查询资料。但是，对于大部分用户来说，还没有办法很方便自如地使用这种形式的浏览器来浏览 Web 页面，因此，在这个阶段，在 World Wide Web 技术发展的道路上就出现了一个鸿沟。当然，后来随着 Mosaic、Netscape、IE 等新一代的基于图形界面的浏览器的出现，大大降低了使用 World Wide Web 技术的复杂度，使得用户能够轻松地通过点击各种超链接来访问 Web 页面，于是大家都开始使用这一技术，从而成功地跨越了这一鸿沟（如今 WWW 服务已经成为 Internet 上发展最为迅速的服务之一）。

从这个例子可以看出：如果技术本身太复杂，无法让人接受，就会导致鸿沟的存在。

稍后我会讲述如何跨越技术发展过程中的鸿沟。在此之前，我先给大家举下面的一些例子。这些例子都是我曾经研究过的，可以说我曾深受其害，因为其中有许多鸿沟都是目前暂时无法跨越的，而我当时没有看到这一点，而花了大量的时间和精力去研究它们。

1. 模糊逻辑（Fuzzy Logic）

1990 年我在日本东芝公司的时候，当时最热门最流行的概念就是“模糊”。几乎所有的电器都被加上了“模糊”的特性。商家在宣传自己的电饭煲、热水器、洗衣机、冰箱、空调、电视机等电器时，都会标榜自己的产品是“模糊”的。但是，如果你现在去日本，就会发现人们不再使用“模糊”的概念了。因为后来人们在实践中发现，“模糊”的概念其实并没有用，并没有像炒起来时所说的那样神



奇，“具有智慧和智能，能自动满足人们的需求”，因而大家逐渐对其丧失了兴趣，于是模糊技术就开始没落了。

2. 遗传算法（Genetic Algorithm）

这个领域从 20 世纪 60 年代开始就十分热门。但是我认为遗传算法这一技术目前连早期接受者这一阶段都还没有到达，还仅限于先行者这一阶段。虽然至今已经有很多关于这个概念的文章发表，但是几乎所有的工作都集中在学术研究领域，很少有实用的东西出现。当然，目前有一些大公司和大学正在研究。我认为将来它的确会有一个很大的市场。

3. 神经网络（Neural Network）

这个领域经历过几起几落。神经网络是在 20 世纪 50 年代开始兴起的，几年之后，麻省理工学院教授马文·明斯基（Marvin Minsky）发表了一篇文章，表明了神经网络的不可行性，于是神经网络技术逐渐受到冷落；后来到了 20 世纪 80 年代，神经网络又重新热门起来。当时学术界普遍认为：神经网络可以模拟人脑的功能，通过学习，神经网络可以完成人脑能够完成的大部分功能。于是人家都非常兴奋，甚至有人还乐观的估计：只要能够在计算机上生成像人脑一样多的神经元，就可以达到使用神经网络模拟人脑的目标。但是人们忘记了一条基本原则：人脑固然是由众多的神经元所组成的，但是人脑的智慧更重要的是依赖于信息在众多神经元之间的传递方式。由于无法掌握这一原理，神经网络遇到了一个鸿沟，因此在发展了十几年以后，神经网络又再度陷入困境。目前神经网络仍然没有在日常生活中得到广泛的应用，而仅局限于某几种应用。





总结

上述这些技术的共同点是：它们看起来都是非常先进、非常吸引人的技术，但是这些技术本身太复杂，需要许多高深的知识才能使用，一般的用户难以接受，因而难以得到广泛应用。

第 3 章



从研究到产品

3.5.2 跨越鸿沟

那么如何跨越这个鸿沟，使产品顺利跨过早期接受者阶段从而走向早期消费群体阶段呢？

杰夫·摩尔在他的 *Crossing the Chasm* 一书中使用“保龄球道（Bowling Alley）”这个很形象的比喻，形容出现鸿沟—跨越鸿沟—进入成熟期这一过程，如图 3.3 所示。

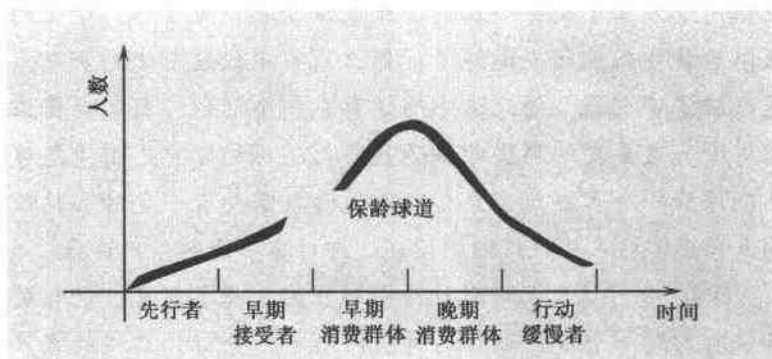



图 3.3 保龄球道

为什么会用保龄球道来形容这个过程呢？打过保龄球的人都知道：在打保龄球的时候，我们并不是用手中的球把所有的瓶子都打倒，而是先打倒 2 到 3 个瓶子，再用这 2 到 3 个瓶子去撞倒旁边的





几个瓶子，然后这些瓶子再去撞倒更多的瓶子。同样，我们在跨越鸿沟时就需要采取这样的策略。

因此，跨越鸿沟的关键在于要控制好保龄球道的策略，使产品成功地打入市场。具体来说必须实现以下两点。

1. 提供一个全面的解决方案

前面我们提到，神经网络虽然没有在日常生活中得到广泛的应用，但是仍然有几家公司在神经网络应用上取得了成功，如美国的HNC公司。

案例一

在美国，信用卡是非常普及的，商家非常鼓励人们使用信用卡来借钱花费，这样商家就可以从中赚取高额利息。但是商家在给客户发放信用卡时也要冒一定的风险。因为如果把信用卡发给了某一个客户，结果该客户花光了信用卡上的钱以后就宣布倒闭或逃跑了，那么商家不仅没有赚到利息，反而赔了不少钱。因此这中间就存在一个问题：商家要保证信用卡一定要发给那些能够还钱、信誉好的客户，但是怎样才能评价一个客户是否合格呢？这时就需要有一定的咨询机构来考察哪些人借钱后能够偿还，即对客户进行信用评估。

HNC公司从信用评估这个领域出发，将各种客户的相关特征输入到一个专门设计的神经网络中，分析什么样的客户最符合要求，从而在个人信用评估领域获得了成功。之后，HNC公司又采用保龄球道的策略，借助在个人信用评估领域的经验将神经网络应用到其他类似的领域中，如公司或企业的信



用评估，同样取得了成功。正是使用这种从个人到企业的全面的解决方案，HNC 公司成功地跨越了神经网络的鸿沟，使该技术在信用评估市场上占有一席之地。

再如，目前许多大公司都在使用 ERP（Enterprise Resource Planning，企业资源计划）系统。ERP 系统是整合了企业内部和外部的所有资源，使用信息技术建立起来的面向供应链的管理工具。借助信息技术，使企业的大量基础数据共享，以信息代替库存，最大限度地降低库存成本和风险；并借助计算机，对这些基础数据进行查询和统计分析，提高决策的速度和准确率，改变落后的企业管理模式，进而建立起一套新的符合市场经济体制的企业管理模式。因此，ERP 系统既非常复杂，也非常昂贵。为了让如此复杂、昂贵的系统能够吸引足够多的用户，ERP 零售商通常都采用这样的解决方案：在销售 ERP 软件系统的同时，为用户提供全套的技术支持和服务。

案例二

例如，假设一个汽车公司购买了某 ERP 零售商的 ERP 系统，那么该 ERP 零售商不仅会提供 ERP 软件，而且同时还会派出许多技术人员进驻到该汽车公司。这些技术人员不仅精通 ERP 软件的使用，还非常了解汽车行业的知识，他们会对汽车公司的员工进行严格培训，以保证用户可以正常地使用该 ERP 系统。如此一来，汽车公司就会觉得这套 ERP 系统使用起来非常简单，他们完全不用做任何事情，因为所有的事



情都被 ERP 零售商做了，汽车公司只需尽情地享受该 ERP 系统为本公司带来的巨大利益和方便。

这样，该 ERP 零售商就会很快在汽车行业取得成功。事实上，目前有相当多的汽车公司已经采用了 ERP 系统。接下来，该 ERP 零售商就可以采用保龄球道的策略，利用在汽车行业所获取的经验，逐步渗透到金融、信用、飞机等行业，从而占领更多的市场。

2. 使客户感受到可靠

要在一个复杂的环境中帮助用户找到一个切实可行的实施方案，在众多繁杂的数据中整理出最有用的信息。

案例三

以我研究的老本行——语音识别系统为例。语音识别系统自诞生以来，由于各方面的原因，现在还没有得到广泛的接受。许多计算机用户仍然选择使用键盘、鼠标进行输入而不愿意使用语音识别系统进行输入。尽管如此，语音识别系统在美国仍然销售得很好。真正广泛使用语音识别系统的用户集中在医生和律师。通常，这些医生和律师都需要书写大量的报告，但是他们的时间非常有限，打字速度也不快，并且平时使用的都是一些非常专业的词汇，因此书写报告是比较费时的。

在这种情况下，有的公司就看到了其中的潜在市场。由于医生和律师的专业术语相对比较固定，这些公司就专门设计





出一套针对他们的语音识别软件，在软件中加入了所有的专业术语，并对软件进行优化，使得语音的识别率非常高。而且，为了保证以后出现新的术语时，这套语音识别软件也能够识别出来，这些公司还专门设计了相应的模块来实现新增术语的记录和添加。然后，公司就主动上门向医生和律师推销这套语音识别软件，并帮助他们进行训练。经过半个小时的适应性训练以后，这些医生和律师就发现这套软件非常好用，可以大大提高他们书写报告的效率，从而节省大量的时间，给自己带来极大的便利，于是他们就会购买这样的软件，从而使得语音识别系统在这两个行业中得到了广泛的应用。

正是由于这些公司真正深入地了解了这两个行业的市场需求，给用户提供了—套切实可行的解决方案，才使自己的产品获得了成功。

3.5.3 龙卷风期

我们称进入成熟期后的这段时期为龙卷风期，如图 3.4 所示。在这个阶段，产品的推广处于疯狂的增长过程中，非常多的用户热切希望使用这个产品。在达到这一阶段后，要注意此时产品的推广方法与以前应该有所不同。这个阶段的产品方案必须考虑为众多用户所设计，即要注意以下几点。

(1) 不仅要尽可能地使产品更加成熟，更重要的是使产品变得更加易于使用。这个阶段的产品必须十分实用，当你有一个很先进的技术时，不要立即投入，因为这通常会使你的产品变得十分复杂，以致大部分用户无法成功使用，必须将其化繁为简。例如，微软的 FrontPage 就是一个在这方面做得很成功的产品。FrontPage 的功能非



常强大，它为企业、个人用户等提供了多种制作主页的功能，但是它的使用方法却非常简单，用户通常只需借助帮助文档，即可轻松地使用 FrontPage，而无须他人的指导。

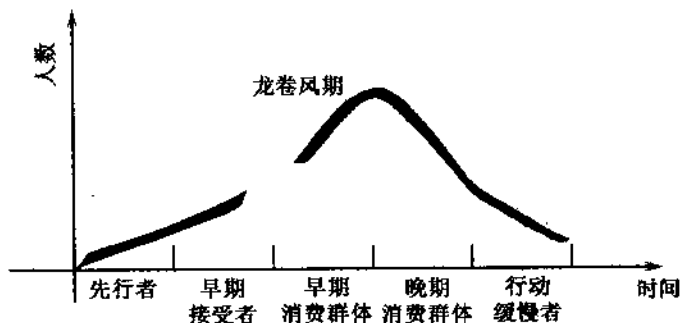


图 3.4 龙卷风期

(2) 要集中精力推广使用此技术，使其尽可能地被广泛使用，争取尽可能多的用户。例如，用此技术做出各种各样很成功的案例，让广大的用户都可以参考使用。而不要集中去做一个或几个功能非常强大但适用性不强的案例。

(3) 争取使此技术成为市场的标准。通常，在这个阶段，市场也会希望某一项技术能成为标准，以便于市场的规范统一和兼容性要求，同时也便于用户之间的知识共享和交流。这样一来，就会有更多的人使用这项技术，形成滚雪球效应。

(4) 如果产品能够在龙卷风期成为市场的领导者，那么从长期来看，市场就基本上被该产品占据了。也就是说，此时的胜利就代表了今后长期的胜利，以后其他同类产品要想打入这个市场是非常困难的。



3.6 基本规则

最后我介绍一下根据自己多年的实践所得到的从基础研究到产品制造的三大基本规则。

3.6.1 寻找市场最需要解决的问题

不要沉溺于先进的技术，而应该去寻找那些市场最需要解决的问题。即把精力集中在人们遇到的那些燃眉之急的问题上。当人们遇到这些问题正在寻找解决方法时，如果你及时推出了一种理想的解决方案，即使技术上还很不成熟、产品还不实用，人们也会对该解决方案寄以厚望，甚至会多付钱或者提前付钱来帮助你，以便你能最快地研制出新产品。反之，如果你研究某一项技术只是因为对它感兴趣，而不去考虑人们的实际需要，那么即使你研究出来的产品非常好，人们也很有可能对它根本不感兴趣，这样的产品自然就会失败。

案例

例如，我在 Nuance Communications 公司研究过的电话语音系统 IVR（Interactive Voice Response，自动交互语音响应）就是一个非常成功的例子。当时（1996 年）美国股票市场的行情非常好，许多人都在炒股，经常有非常多的客户给股票证券公司打电话咨询股票行情，因而应接不暇。如果没有一种很好的解决办法，股票证券商就不得不专门新盖一栋新的大楼，然后雇佣成百上千的员工，对他们进行培训，并集中在大楼中充当接线员，以便能够及时地为众多的客户提供应



答服务。由于接线员的工作非常枯燥，因此他们流动得非常快，股票证券商每年都需要重新雇佣大约 50% 的员工，重新对新员工进行培训，这对公司的管理也造成了很大的问题和压力。

后来，股票证券商们看到了 Nuance Communications 公司的 IVR 方案，感到非常满意，简直就像找到了救命草一样，马上宣布要购买 IVR 系统。当时 Nuance Communications 公司的 IVR 系统还没有完全实现，这些股票证券商们仍然迫不及待地提前付了钱，希望能尽快使用 IVR 系统。之所以会出现这种买家围着卖家转的情况，就是因为该公司寻找到了股票交易市场最需要的一个平台。一使用 IVR 系统以后，用户打电话咨询股票行情时，只需根据电话中的电脑语音提示来进行语音输入，即可得到答案，而不再需要大量的接线员，从而为股票证券商节省了大量的人力、物力和财力。

后来，IVR 系统在其他许多领域中也得到了广泛的应用，目前 Nuance Communications 公司已经成为美国领先的一家从事电话语音系统业务的公司，它的成功正说明了找对问题的重要性。

3.6.2 由小处着手，从全局思考

不要总是想着：“我要发明一种技术，它会改变全世界！”比如让全世界的交通系统都因为我的技术而改观。事实上，即使研究出了这样的技术，也将非常难以推广。一个很好的例子就是人们已经研究了许多年的电汽车。即使现在电瓶的存储能力提高了，但是仍然存在许多的问题，如在电瓶的电即将耗尽时没有办法方便及时地



对电瓶进行充电，除非建造许多类似于现在的加油站一样的充电站。这在目前显然是不可能的。为了使用一项新技术，而导致整个产业或社会的基础设施的类型都发生改变，这是极其困难的。除非是政府下定决心耗费巨资对基础设施进行改造，否则仅依靠一两家公司力量来推广这项新技术将难如登天。

因此，在刚开始进行研究的时候，要着眼于去解决一些比较小但又是非常迫切的问题。同时还要有长远的眼光，不要因为解决了针对一小群人的迫切问题之后就满足了而停步不前，还要想办法去解决另一小群人的迫切问题，然后慢慢地延伸并扩散到各个族群，逐步提供大多数人所需要的功能。

所以对任何人而言，产品的推出要循序渐进，从小处着手，从全局思考，而不能过于急躁。这是解决问题的核心。

案例一

以色列的 Checkpoint 公司是一家由两个年轻人组建的公司。刚开始他们只是做一些防范黑客（Hacker）入侵网络系统的产品。在取得了成功之后，他们逐步渗透到网络安全的各个方面，到目前为止，Checkpoint 公司已经成为世界上最大的防火墙公司。

案例二

英国的 Autonomy 公司是由剑桥大学一位博士生创办的。他利用机器学习和数据挖掘技术为人们提供了如何更快地在互联网上寻找信息的方法，到如今，Autonomy 公司也发展成了市值达几十亿美元的大公司。



案例三

Firefly 公司是由麻省理工学院多媒体实验室的一些研究生成立的。这些年轻人把很多人的不同兴趣整合在一起，然后采取一些技术手段对这些兴趣进行分析，最终可以根据一个人在某一方面的兴趣来猜测他在其他方面的兴趣。就是这样的一个将消费者进行聚类的产品，最后被微软收购。

从这三个例子可以看出，尽管这些公司都是一些年轻人创建的小公司，但最后都取得了巨大的成功。

3.6.3 品牌并不代表全部

很多人认为品牌就代表了财富，如果拥有了名牌，就等于拥有了成功。事实并非如此。即使是很有名的品牌，也并不一定代表价值。品牌必须具备深厚的基础，能够提供实际的价值，那么它才会变得有价值，使人们依赖它。例如，可口可乐（Coca Cola）、迪斯尼（Disney）、麦当劳（MacDonalds）等都是货真价实的品牌，它们都有深厚的基础为保障。相信大家都喝过可口可乐，可口可乐公司能够保证在全世界的任何地方，消费者购买的可口可乐的味道基本上一样。这一点看起来很简单，其实这并不容易做到。因为各地的水质、原料都不一样。这绝不是仅仅靠外表的包装就能做到的，起关键作用的是可口可乐公司强大的销售系统。同样，迪斯尼和麦当劳在各地的服务和形象都倚仗一套统一的人员训练方法来维持。所以我们研究出来的产品必须有真材实料，能够得到广大用户的认可，否则即使靠打广告或媒体炒作创造出了一个品牌，恐怕这个品牌很快就会一文不值了。

3.7 推荐书目

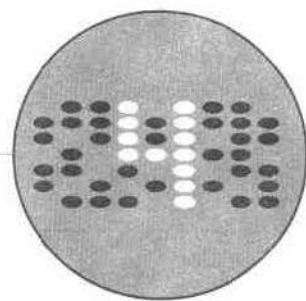
最后，我推荐一些非常好的书，建议大家去阅读一下：

- 、 Crossing the Chasm (Geoff Moore)
- 、 Into the Tornado (Geoff Moore)
- 、 The Gorilla Game (Geoff Moore)
- 、 The Macintosh Way (Guy Kawasaki)
- 、 Rules for Revolutionaries (Guy Kawasaki)

第 3 章



从
研
究
到
产
品



第4章

微软的软件开发

Software Development at Microsoft

背景

现在的软件开发不再是个人英雄主义打天下的时代了，尤其是像微软这样大的软件公司，一个软件都是由几百人甚至几千人共同合作完成的。那么如何管理这样庞大的开发阵容？员工是如何分工的？他们之间又是如何协作的？这些都是大家关心的问题。陈宏刚博士结合自己在微软公司的亲身体会，并结合具体实例，从一个较高层次介绍了微软的产品团队、软件开发过程和开发方法。

微软亚洲研究院商务及高校关系高级经理,主管微软亚洲高校关系、微软亚洲研究院人力资源和商务计划及发展等。云南大学、同济大学客座教授。

1982年 获兰州大学计算数学学士学位。

1987年 获西安交通大学计算数学硕士学位。

1987年 赴美留学,后来获华盛顿大学应用数学博士学位。

曾在西安交通大学工作五年半。获得博士学位后,在华盛顿大学做博士后研究。

1995年加盟微软公司,在微软总部先后做过测试工程师 (Software Test Engineer)、测试组长 (Test Lead) 和测试经理 (Test Manager),参加过微软产品 Windows 95, Exchange Server 4.0 和 4.5, Internet Explorer 4.0 和 5.0, SQL Server 2000 的开发和测试。通过四年的亲身实践,摸索了许多宝贵的经验。

陈宏刚博士在微软亚洲研究院办公室门前



陈宏刚

本章内容概览

- I 概述
- II 微软的产品团队
- III 微软的软件开发过程
 - I 想法和意图批准里程碑
 - II 产品计划的通过里程碑
 - III 范围完成/第一次使用里程碑
 - IV 发布阶段

4.1 概述

软件开发是一个非常复杂的过程，特别是像微软所开发的大型或超大型软件产品。本章对微软软件开发过程作一个大概的介绍。在介绍微软的软件开发过程之前，首先简单介绍一下微软的组织架构和新产品的产生过程。

4.1.1 微软的组织架构

从公司的组织结构来看，微软设立了很多部门，包括市场营销部门（主要做销售、市场、售后服务等）、内部营运部门（负责公司内部的基本管理等）、产品开发部门（公司赖以生存的最主要的部门，负责所有软件的开发）；以及研究部门（进行基础研究和新技术研究）。这些部门中又包含了许多子部门。设立这些部门的目的是为了合理分工、有效管理、提高效率。

4.1.2 新产品的产生过程

在微软，软件产品的研发由产品开发部门及研究部门负责。每个新产品从无到有的产生过程（New Product Decision Process）一般都要经过以下几个开发步骤。

（1）新产品项目的提议

如果要开始一个新产品，首先需要对新产品进行提议。提议者可以是我们的用户，也可以是我们的管理部门或者是我们的工程师。

（2）市场分析预测

提出一个新产品项目后，接下来要做的事情就是进行市场分析预测。主要任务是判断这一新产品是否有市场，是否是大家需要的产品。

（3）技术可行性分析

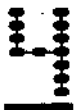
在市场分析预测的基础上，我们需要研究确定实现这个新产品在技术上是是否可行。只有技术上能够实现时，我们才能决定去做该产品。

（4）产品研发实施步骤

在技术分析论证通过之后，我们就开始了具体的、详细的产品研发和实施步骤。

（5）高层论证和审批

当具体的产品研发计划写出来后，就需要进行高层论证和审批，争取支持者。如果要开发一个新产品，必须要找到一群支持者，否则，新产品是无法开发的。但是，也有一些幸运儿，他们有非常好的信誉，所以通常就可以很容易地通过这一步骤。



案例

我以前在 SQL Server 团队时的老板就是一个非常有信誉的人。在微软总部，他是出了名的创新者，所以，他提出的任何新产品都会很容易被批准，高层根本就不需要论证。比如，开始他想做面向对象（Object Oriented）的产品，公司马上就同意了，结果他很快就做出来了。现在，我们许多技术都用上了他的新产品：过了不久，他又想做 Trident（产品的内部名称），准备把许多对象用到一个新的方法上，于是他就带领一个团队进行开发。后来他把所开发出来的产品用在了 IE 上，成为动态 HTML（Dynamic HTML）和 ActiveX Control 这些技术；当这些技术开发完毕后，他突然又离开了，去一个新的团队领导开发 XML。当时 XML 还没有被业界重视，他带领团队进行 XML 开发的时间比 XML 国际标准的公布时间提前了两年。当时也是因为他非常有信誉，所以公司马上为他提供人力和财力，让他立即开始进行开发。正是因为他，才使得我们公司不仅没有在 XML 这个领域落后，而且还是最早参与 XML 国际标准制定的公司之一。

（6）项目确立和执行

获得审批和支持后，就会得到人力和财力支援。这时，就需要确立项目，将项目中的各项工作配置好，然后就可以按照计划执行了。

（7）产品开发

当上述过程都准备完毕后，就可以开始新产品的真正研制和开



发过程。

微软的优良传统

在微软公司有一个优良的传统，就是任何人都可以对产品的研发提出合理化建议。这些建议可以来自于管理人员，也可以来自于一般工程师，更可以来自于一般用户。例如，在微软总部，有些会议室的墙上经常贴了许多小纸条，上面写着大家提出的建议。有关部门根据这些建议进行市场分析预测，分析技术的可行性，制订出产品研发计划后报请上级部门批准。上级部门根据支持者进行分析和认证，确定需要投入的人力资源和财力资源。因此，开发一个软件需要经过相当多的步骤，投入巨大的人力、物力和财力。可以说，微软开发一个软件所花的钱绝对不少于英特尔公司开发出新一代 CPU 所花的钱。

4.2 微软的产品团队

在介绍微软的软件产品具体开发过程之前，首先需要了解一下微软的团队模型（Team Model），也就是微软的一个产品开发团队的组成及其内部人员的分工和职责等情况。通过这种了解，有助于加强大家对微软软件开发过程的理解和吸收。

4.2.1 微软的团队模型

1. 微软的产品团队原则

微软的产品团队始终坚持以下原则：

- ❏ 小型并具有多功能的团队
- ❏ 角色互相依赖并分担责任
- ❏ 具有深厚的技术和商业敏锐性
- ❏ 关注于完成产品
- ❏ 有明确的工作目标
- ❏ 用户积极参与
- ❏ 共享产品远景
- ❏ 人人参与设计
- ❏ 努力从过去的产品中学习
- ❏ 共享产品总体管理和决策
- ❏ 所有团队成员都在同一个地方工作
- ❏ 大团队的工作方式类同于小团队

“People are most productive working in small teams with tight budgets, time deadlines, and the freedom to solve their own problems. (人们在那些具有严格的预算和时间限制，并且能够自由地解决他们自己问题的小团队中工作时将更富有生产力。)”

——比尔·盖茨

在微软的产品团队中，权威仅仅来自于知识，而不是来自于职位。





2. 不同角色的团队

微软的产品团队由一些地位平等的小团队组成，这些小团队在整个产品团队中扮演着互相依赖、互相合作的不同角色，每一个角色都有各自特定的任务。他们共享对产品的管理，也共享对产品的责任。每一个角色都始终存在并作用于整个产品开发过程。图 4.1 所示就是微软的产品团队的组成。

微软的产品团队之所以由一些地位平等的小团队组成，是基于以下考虑：

- ◀ 可以充分根据产品的特点进行权力分配
- ◀ 可以共同分担责任和义务
- ◀ 彼此地位等同
- ◀ 可以听取多数人的意见
- ◀ 可以达到彼此相互制约，最终达到合理的平衡的目的

因此，在整个产品团队的运作过程中，这 6 个小团队之间的互相沟通是非常关键和重要的。

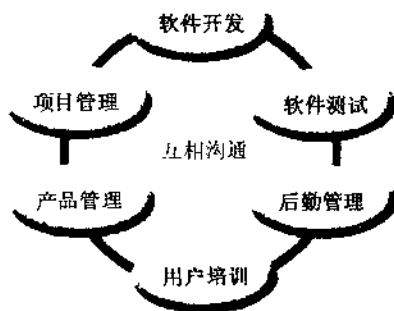


图 4.1 微软产品团队的组成

表 4.1 中列出了各个团队的角色及其主要目标。

表 4.1 各个团队的角色及主要目标

团队角色	主要目标
产品管理	确定产品的远景，获取并确定用户的需求，开发并维护商业安全，满足用户的需求
项目经理	制定开发功能规范，在团队内进行沟通和协调，维持产品进度并报告产品状态，保证产品尽快尽好地在产品约束条件下发布产品
软件开发	开发出满足设计规划于用户要求的产品
软件测试	开发测试策略和计划，保证在解决了所有已知问题后再发布产品
用户培训	保证使用文档要求全部很清楚地写出来，提高用户使用产品的技能，保证大多数用户都能够充分利用产品的功能
后勤管理	保证产品能够平稳地发展

整个产品的管理是由一个高级的产品管理团队来完成的。这个团队是由各个团队的主要负责人组成的，如图 4.2 所示。该团队负责计划和安排所需的资源，进行风险评估，并对产品计划进行跟踪。

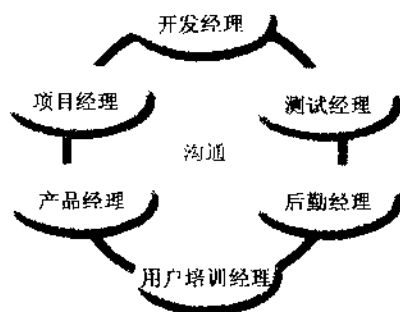


图 4.2 高级的产品管理团队的组成

3. 小团队

有时候，微软更愿意使用规模较小的产品团队。这是因为小团队具有以下优点：

- 可以减少因为互相沟通所造成的消耗
- 可以降低开发过程的消耗



- ✧ 可以降低管理的消耗
- ✧ 可以更快速地完成产品
- ✧ 可以保证更高的产品质量

在开发一个小的产品时，我们就可以考虑使用小的产品团队。通常对产品团队进行压缩的方法就是将图 4.1 中所示的各个角色进行互相组合。但是，并不是所有的角色都是可以合在一起的，表 4.2 中详细列出了各个角色之间可能的组合情况。

表 4.2 各个角色之间可能的组合情况

	产品管理	项目管理	软件开发	软件测试	用户培训	后勤管理
产品管理		N	N	P	P	U
项目管理	N		N	U	U	P
软件开发	N	N		N	N	N
软件测试	P	U	N		P	P
用户培训	P	U	N	P		U
后勤管理	U	P	N	P	U	

其中：P 表示可以组合，U 表示不太可能组合，N 表示绝对不能组合

从表 4.2 中可以看出，软件开发的地位是非常重要的，这个角色是绝对不能和其他任何一个角色组合在一起的。软件开发团队的任务就是要专心致志地进行程序开发，而不能分心去干其他的工作。

例如，我们可以将产品团队压缩成如图 4.3 所示的样子，此时只有三个团队了，除了软件开发团队没有动以外，项目管理团队和后勤管理团队合成了一个团队，而软件测试团队、产品管理团队和用户培训团队合成了一个团队，这样整个产品团队只包含了三个小团队。

总之，微软的团队模型具有以下优点：

- ✧ 每一个团队成员都与产品的成败有直接的利害关系
- ✧ 可以创造一种鼓励简单、清楚、高效、合作、承诺和团队合作精神的的企业文化
- ✧ 提高每一个团队成员的责任感和义务感

- ✎ 可以使产品能够着眼于用户的需求来进行开发

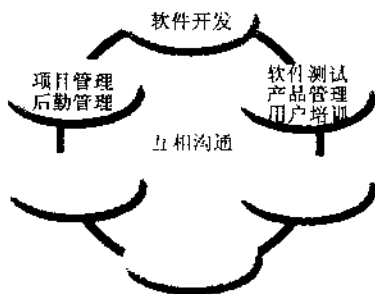


图 4.3 将产品团队进行压缩

4.2.2 微软软件开发的过程模型

1. 指导原则

微软的软件开发过程遵循以下指导原则：

- ✎ 软件的开发过程是由目标驱动的，而不是由具体任务驱动的
- ✎ 开发过程的各个里程碑（Milestones）都是显而易见的
- ✎ 基于版本的发布
- ✎ 进度表是基于风险驱动来制定的
- ✎ 依靠完全的团队合作来完成
- ✎ 严格管理，保证质量

2. 开发过程原则

具体来说，微软的软件开发过程的原则如下：

- ✎ 在制定进度表时，要明白我们要面对的是一个不确定的将来
- ✎ 通过有效的风险管理使不确定性因素达到最少
- ✎ 定期的编译和快速的测试来使产品的稳定性和可用性达到最佳



- 产品的开发周期要快速
- 通过创造性来实现功能的改进而不增加资源
- 建立固定的进度表
- 使用小团队来并行工作，以达到最多的同步点
- 将一个大的产品分割成一些易于管理的、可以在几个月之内完成发布的小部分
- 要避免任何不必要的功能扩展
- 使用概念证明（Proof-of-Concept）的技术原型来做开发前的测试
- 要本着零缺陷的态度（一般来说，一个软件产品不可能没有缺陷，这里的零缺陷是指产品发布时没有影响产品功能和使用的缺陷）
- 在阶段回顾时要集中于功能和产品的改进，而不是挑任何人的过失

4.2.3 产品开发团队的主要组成部分

下面对微软的产品开发团队的主要组成部分做一下具体介绍。

1. 产品管理团队（Product Management Team）

这是产品的管理部门，管理部门负责人即为产品总经理（General Manager，简称 GM）。产品总经理下面会有一个或者几个产品单元经理（Product Unit Manager，简称 PUM）。同时产品管理团队中还包括产品计划、市场分析和研究、产品推销和公共关系的负责人。

图 4.4 所示就是产品管理团队所承担的角色。

从图 4.4 中我们可以看出，产品管理团队承担的角色主要是产品团队的管理（Group Product Management），包括：产品计划（Product Planning）、市场分析和研究（Market and Research）、产品推销（Evangelism）、市场（Marketing）、公共关系（Public Relations）等。

具体地说，产品管理团队承担的角色如下：

- 了解用户的想法
- 确定项目的发展前景

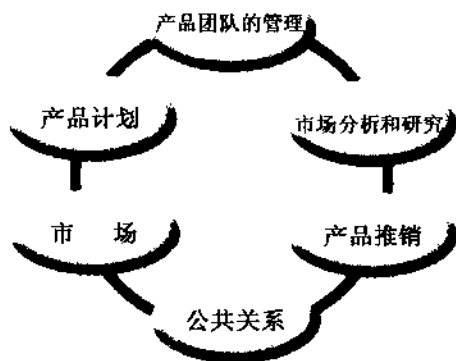


图 4.4 产品管理团队所承担的角色

要确保新产品的功能能够实现。正如前面所说的，首先需要提出新产品项目的提议，接着需要进行市场和技术可行性分析，从而确定需要做的产品，以及将要达到的目标。

■ 需要清楚地知道用户的需求，并详细地给出定义

因为很多时候用户只是口头问我们能不能做出具备某种功能的产品，并不能很清楚地说出他的需求，例如，用户可能会说：“你们能不能开发一种新产品，以便计算机能够听得懂我说话？”仅从他的这句话是很难定义用户对新产品的需求的，所以，我们必须经过一定的转换，结果可能就是：用户需要一种语音识别（Speech Recognition）软件。因此，产品管理团队必须将用户的需求清楚地定义出来。

■ 我们要确保新产品能够为我们带来利润

任何产品最终还是要以商业利润为最终目的的。



■ 控制用户的期望值

一方面我们要让用户知道我们能够做到的功能，但是不能让他们抱有太多的期望。这一步的目的是让用户体会到我们的确是在按照他们的需求做这件事，但还不知道最后的结果是否一定能满足他们的需要。所以，产品管理团队需要控制用户的期望值，不能使他们的期待值太高，事实也的确如此，在没有十足的把握以前，没有哪个人能把今后的事情说得很清楚。

■ 设计产品的特性和进度表

产品管理团队需要设计新产品的特性（Feature）及开发进度表（Schedule）。当然，这时所需要的只是一个大概的特性定义和进度表。产品管理团队并不制定具体的产品特性和进度表，但他们知道大概需要达到的目标。同时，还要做一些折中决定（Trade Off Decision），也就是说，如果产品的特性中存在一些冲突的问题，就必须根据新产品最终要达到的目标，决定要取舍的内容。在本章的后半部分将详细介绍这种折中决定。

■ 负责管理市场、推销以及公共关系

产品管理团队要让所有的人都知道我们将做的新产品，以及该产品将具有的功能，将为人们提供的帮助等。

2. 项目管理团队（Program Management Team）

项目管理的任务是控制决策的各种因素，以保证在合适的时间推出合适的产品，同时负责创建功能规定文档，并将它作为如何实施产品或服务的一种决策工具。最后，项目管理将面对使产品或服务与组织标准和操作目标相一致的日常协调工作。

图 4.5 所示就是项目管理团队所承担的角色。

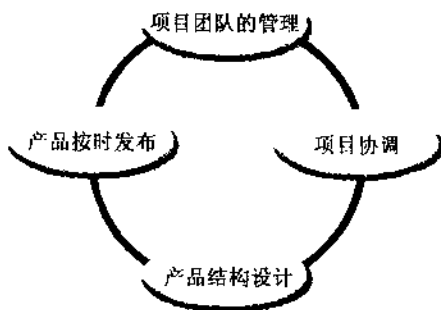


图 4.5 项目管理团队所承担的角色

从图 4.5 中我们可以看出，项目管理团队承担的角色主要是项目团队的管理（Group Program Management，简称 GPM），包括：项目协调（Project Coordination）、产品结构设计（Product Architecture），以及产品按时发布（Release Management）等角色。

具体地说，项目管理团队承担的角色如下。

■ 操纵着产品的开发过程

确保我们是按照产品开发进度表一步一步地进行的。

■ 管理产品的细节

将产品的细节写出来，详细地说明我们需要的具体内容。例如，不能只是说我们要做语音识别，而需要一步一步地写清楚“语音识别怎么做？”，“如何完成？”等等。

■ 促进团队内部的交流和商议

产品项目经理需要与开发人员、测试人员进行交流和商议，大家可能会讨价还价，从而确定真正能够实现的东西。

■ 保证产品的开发进度

产品的开发一定要满足我们的进度表。但是有一点大家都可能感觉得到：软件产品经常要推迟发行。许多做软件的公司都会把软



件开发时间和产品发布时间说得比较宽裕些。例如，如果我有一个开发期限为两年的新产品，预计明年 1 月份要发布，但我会对别人说：“明年 4 月份发布。”即使是这样，实际发布新产品的时间可能还要推迟。

推迟的原因很简单，因为我们很难把握软件的开发过程。在软件开发过程中，许多事情是超出我们控制的，经常会有一些预料不到的事情发生。其中不仅有技术上的问题，也有其他方面的问题。如用户提出了新的要求，或者在实现过程中出现了一些问题。凡是做过软件，特别是做过大型软件开发的人都应该能够理解和明白这一点。所以，项目经理的一项任务就是尽量保证我们的产品能够按时完成。有时为了保证产品的进度，项目经理需要做出一些取舍：只要不影响用户的使用，宁愿舍去一些功能，也要赶时间。其实，每一个软件公司都有推迟发布产品的现象，只不过由于微软的产品用得最广，有时候影响比较大。但是，现在用户可以发现，微软产品发布的推迟现象越来越少，基本上都是按时发布的。这是因为我们的经验越来越多，基本上可以准确地预计出产品能够发布的时间。

■ 控制全局，并做出一些折中决定

在产品开发过程中，项目经理还需要做一些折中决定。例如，测试人员认为必须更正某 Bug，但开发人员认为根本不需要更正此 Bug，而且如果要更正的话，风险很大，会影响所有的代码。此时，往往项目经理就需要与开发人员和测试人员一起进行讨论，然后根据实际情况做出折中决定。

3. 软件开发团队（Software Development Team）

开发人员的任务比较单一，主要就是负责代码的设计和程序的实现。

图 4.6 所示就是软件开发团队所承担的角色。

从图 4.6 中我们可以看出，软件开发团队承担的角色主要是开发管理（Development Management），包括：建立一个很好的数据库

（Database）、系统服务（System Service），以及用户界面（User Interface）

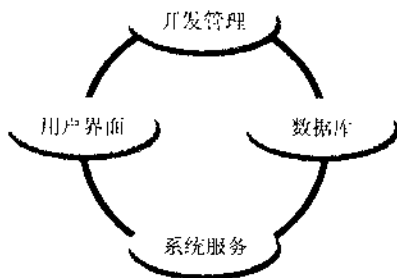


图 4.6 软件开发团队所承担的角色

具体地说，软件开发团队承担的角色如下。

■ 具体说明物理设计的功能

开发的程序能够满足所有的物理设计要求，这是开发人员的主要责任。

- ✎ 估计完成每一个功能所需要的时间和人力
- ✎ 建立一个很好的开发数据库

任何产品都可能涉及到数据库开发，所以一定要有数据库，该数据库包括了开发人员的程序库。

- ✎ 编码实现并建立功能
- ✎ 为配置准备产品

4. 软件测试团队（Software Testing Team）

软件测试团队的任务很清楚，就是站在使用者和攻击者的角度上，通过不断地使用刚开发出来的软件产品，尽量多地找出产品中存在的问题，也就是我们所称的 Bug。图 4.7 所示就是软件测试团队所承担的角色。

从图 4.7 中我们可以看出，软件测试团队承担的角色主要就是测



试管理 (Test Management), 包括: 一致性测试 (Compliance Testing)、配置测试 (Configuration Testing)、集成测试 (Integration Testing) 及强力测试 (Stress Testing) 等。

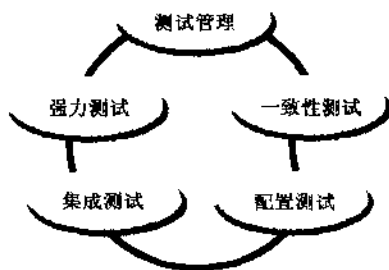


图 4.7 软件测试团队所承担的角色

具体地说, 软件测试团队所承担的角色如下。

- 1、尽量找出所有的 Bug

也就是说, 测试人员要保证尽量知道所有的问题。

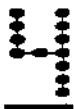
- 2、写出测试规范和测试计划

通过测试规范和测试计划, 测试人员能够告诉别人自己做测试的方法和步骤, 以及保证测试能够找出所有应该知道的问题的测试策略。

4.3 微软的软件开发过程

4.3.1 微软软件开发过程的特点

软件开发的过程非常长, 需要很好的管理。我有好几个朋友自己开软件公司, 但最后都倒闭了。我问他们原因, 他们说是在开发人员那里出了问题。为什么呢? 因为核心的开发人员走了,



项目就不知道怎么继续了。这就是管理的问题，说明他们的管理方式很糟糕。微软的团队非常大，通常都有几百到几千人，在开发过程的中途，也经常会出现开发人员离开现象。但是，这对我们产品的开发并没有影响，根本原因就在于我们在软件开发管理上做得很好。

根据我的经验，微软的软件开发过程具有以下几个特点。

- 第一点，也是最重要的一点，在开发任何一个新产品时，微软的所有文档都是非常齐全的，项目的规范也很清楚，所以，每一个开发人员都很清楚自己需要做的任务。一般来说，一份功能规范是由一个小组来做的，每个小组由三到五人组成。根据项目的不同，有的小组可能大一些，也有的小组可能小一些。这样，任务被划分得很明确，每个人都负责自己的一部分。
- 第二点，开发人员之间需要互相阅读其他人新编写的代码。因为大家都是为了实现同一个特性，只是各人分别实现不同部分而已，而且互相之间可能存在调用关系，所以，他们之间是互相依赖的。于是，同一小组的开发人员就必须互相阅读代码，以便相互之间能够非常了解。这样，产品就很难由一个人控制。我们公司绝对不会做这种事情：一个产品的核心全部由一个人控制！否则，如果他一走，大家就完蛋了。
- 第三点，所有的代码都需要有清楚的注释。例如，需要注释下一段实现的算法名称及具体内容，或者下一段的功能是调用数据等。所以，我们的产品一般有两种版本：调试（Dev）版本和发布（Ship）版本。

名词解释

Dev 版本就是包括所有的、非常详细的调试信息的程序；
Ship 版本就是去掉调试信息之后的程序。



这样，在产品开发过程中，如果某个工程师离开公司了，新来的人可以马上阅读已有的产品功能规范文档以及开发文档。通常，开发人员除了 Dev 版本的代码之外，还有一个自己专门的文档，里面记载了具体的实现方法。另外，新来的人还可以咨询小组中其他人员，因为大家的東西都是共享的。所以，这就是为什么微软在产品开发过程中，即使一些工程师离开公司了，也不会影响我们工作的原因。

4.3.2 微软软件开发过程的阶段

微软的软件开发过程包含四个主要里程碑（Milestone），每一个里程碑都是一个阶段的终结点。这四个里程碑是客户与项目组之间重要的设计、评估和协调的同步点。如图 4.8 所示。

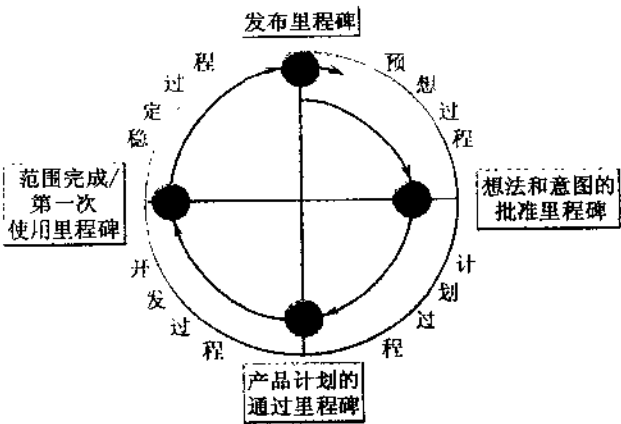


图 4.8 微软的软件开发过程

在产品开发的不同主要里程碑，起主要作用的团队角色也不一样。表 4.3 中列出了在产品开发的四个主要里程碑中所涉及的团队角色。

表 4.3 在产品开发的四个主要里程碑中所涉及的团队角色

产品开发的四个主要里程碑	所涉及的主要团队角色
想法和意图的批准里程碑 (Vision/Scope Approved)	产品管理
产品计划的通过里程碑 (Project Plan Approved)	项目管理
范围完成/第一次使用里程碑 (Scope Complete/First Use)	软件开发 用户培训
发布里程碑 (Release)	软件测试 后勤管理

1. 想法和意图的批准里程碑 (Vision/Scope Approved Milestone)

这是微软软件开发过程的第一个里程碑，其间包括前面说过的：项目提出、市场分析、技术可行性分析三个过程。这一阶段的目标就是努力使新产品的想法和意图得到通过。在此阶段，必须做以下工作：

- 、 陈述项目的目标文档
主要是论述新产品的功能
- 、 评估新产品的风险

新产品的开发会带来各种风险，包括市场和技术上的风险，甚至有可能对我们自己的产品造成风险。这时的风险评估不一定很详细、很准确，但是在项目开始时，也就是在提出新产品建议的同时必须对风险有一个评估。

- 、 描述新产品大概的结构

例如，如果要做“语音识别”这个新产品，那么“语音识别”应该包括哪几部分功能呢？首先，该产品应该将那些录下来的声音通过很多识别转换，变成文字内容；接着将这些文字转换为可编辑的内容……虽然在这里产品的功能说明的不是很详细，但是描述了新产品的大概结构。

2. 产品计划的通过里程碑 (Project Plan Approved Milestone)

在微软总部，如果需要开发新产品或更新老产品时，公司会提



供一间很大的、并且没有窗户的会议室。公司所有的雇员都可以提建议，并把自己的建议写在一张小纸条上，然后贴在这个会议室的墙壁上。因此，常常会出现在一间非常人的会议室墙壁上贴满了上千个甚至上万个纸条的情况。这时，许多项目经理就会每天“面壁”收集整理这些建议。这项工作非常花时间，有时可能需要花好几个月的时间才能完成。然后，这些项目经理再去写具体的产品研发计划。通过这个过程，项目经理可以与用户以及各个部门进行交流。通俗地说，就是可以“讨价还价”，互相达成一种平衡。

这一阶段主要是项目经理的工作，此阶段的目标主要如下

◀ 定义功能规范

主要是将新产品的功能规范很详细地写出来。首先需要将新产品分为几个功能部分进行描述；接着需要写出这些功能的特征；然后，描述每个功能的模块组成，并描述每一个模块具体解决的问题。如 Microsoft .Net 产品，它包括几大部分：用户界面、服务……，同时，你必须很清楚地知道这几大部分各自又包括的内容。其中，用户界面又大概分为几个模块：键盘输入、语音输入、手写体输入……，这些都必须写出来。

◀ 评估风险

对每一个新功能的加入都需要评估风险，例如，“语音识别”功能的加入会不会对产品其他功能产生影响？当加入一个新功能时，会不会对产品的老版本产生影响？这些都需要进行风险评估。

◀ 总体计划

在此阶段，项目经理还必须写出新产品开发的总体计划。该计划不是很详细的计划，它是产品一个较高层次上的开发计划。

◀ 总体进度表

同样，在此阶段，项目经理还必须写出新产品开发的总体进度表。

3. 范围完成/第一次使用里程碑 (Scope Complete/First Use Milestone)

这是一个非常重要的阶段。在此阶段，主要是很清楚地定义出所有版本的功能规范 (Versioned Function Specification)。如果所有的代码都完成了，那么进度表也就可以确定了，此时一般都可以正确地估计出产品的发布日期。

这一阶段主要是开发人员的工作，此阶段的目标主要有：

- ✧ 完成全功能代码的编写
- ✧ 风险评估
- ✧ 在线和打印的文档
- ✧ 测试规范
- ✧ 版本化的功能规范
- ✧ 更新的进度表

4. 发布里程碑 (Release Milestone)

软件开发的最后一个里程碑就是代码编制完毕之后的稳定和发布过程。其实上面所讲的软件开发的第三个里程碑就相当于一个 Alpha 版本过程，也就是说，产品还没有完全写完、还没有完全做好。就像搭建一座房屋一样，从开始搭横梁，接着修建房间，再把所有的房间墙都填好，这都是上面第三个阶段需要完成的工作。可是房间修建好之后，为了让人们居住，还必须对其进行一些设计，进行装修和装饰，让房间显得很漂亮，更适合人们居住。以上这些工作就相当于软件开发的第四个阶段，即保证产品能够使用。其实这一阶段就相当于 Beta 版本过程，即可以试用的阶段。主要的目的就是让产品越来越稳定、越来越完善，将所有需要解决的问题都尽快更正。

这个阶段主要是测试人员的工作，经过此阶段之后，我们就可以完成以下工作：



- ✧ 得到最后的产品（Executable Product）
 - ✧ 发布的文档（Release Note）
 - ✧ 版本资源（Versioned Source）
 - ✧ 培训手册（Training Manual）
 - ✧ 产品所有的文件（Documentation）
 - ✧ 产品出现的问题库（Issues and Bug Database）
 - ✧ 包括已经解决的，以及还未解决的问题和 Bug。
 - ✧ 不同设备和平台的安装（Facility and Platform Installation）
 - ✧ 软件/数据的安装/转换（Software/Data Setup/Conversion）
- 下面我将对各个阶段进行详细解释。

4.4 想法和意图批准里程碑

其实，这一阶段的分析并不是很可靠。坦白地说，微软公司的许多产品在此阶段都通过了，甚至也通过了下面的第二个阶段，但是到了第三个阶段，就可能进行不下去了。也有的产品通过了第一个和第二个阶段，第三个阶段也做得不错了，但是这时出现了新的技术，我们发现正在做的产品已经没有什么用处了，只好放弃。大家只看到微软公司产品成功的一面，其实还有很多产品中途就夭折了，花的人力、物力和财力也非常多。可以说，微软开发一个软件所花的钱绝对不少于英特尔公司开发出新一代 CPU 所花的钱。

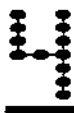
具体地说，在这一阶段我们需要在以下几个方面达成一致。

■ 提出一个长远目标（Long-Range Vision）

例如，我们的长远目标就是让计算机从工具成为我们的助手，让它能听、能说、能看、能想，还能学习。

■ 制定短期的目标（Shorter-Range Scope）

实际上，上面所说的长远目标在短期内很难实现。那么，短期



的目标是什么？就是让计算机能够说话，即具有“语音合成”功能，以及能够听，即具有“语音识别”功能。于是，我们可以先将这两部分功能实现了，然后再一步一步地实现其他功能。

■ 新产品的机会和风险（Opportunities and Risks）

我们要清楚地描述开发新产品的理由，包括它的机会和风险。我们之所以要做这个新产品，当然是希望赚钱，但是该新产品为什么会赚钱？它能够提供什么机会来展示新技术，让人们感觉到你的产品越来越普及实用？这些都必须描述得很清楚。

同时，开发一个新产品也存在许多风险。如果我们的新产品没能开发成功，我们可能就要浪费很多钱；或者某个技术突破不了时，新产品就不能实现，只能中途夭折；或者我们的产品比别人做出来得晚，等等。在这些情况下，我们都可能将全部的人力、财力浪费掉了。所以，必须详细地描述产品的风险。

■ 假设（Assumptions）

我们在做任何事情的时候，开始都会有很多假设。比如：按照现在的速度发展，我们的产品完成时会达到什么程度？假设几年后我们的语音技术越来越成熟，同时，计算机容量也越来越大，那么我们就可以有一个非常好的语音模型，安装以后并不会影响机器的速度……，这些都是假设。在实际工作中，一定要敢于假设。但同时也要注意，假设是一定要有原因和依据的。例如，我们敢假设计算机速度会越来越快，正是因为有摩尔定律的支持；我们可以假设我们的新技术一定会被大家接受，是因为现在的技术还不能满足大家的需求，给大家造成了很多的不方便。所以，假设一定要有根据，而不能靠空想。

■ 约束（Constraints）

当然，我们在决定是否进行一项新的产品开发时，还要考虑到一些约束条件，例如，要考虑财力、时间和可用设备的数量，以及

技术力量是否雄厚等。这些都是做决定时的约束条件。

■ 项目资源 (Project Resources)

如果要进行这项新产品的开发，我们要计算一下需要多少人力的参与和财力的投资。

■ 时间和工作强度 (Time and Effort for Planning Phase)

最后，我们还要估算一下大概的时间和工作强度。

以上就是软件开发的第一个阶段，这是一个总体的过程。这个过程很粗略，所以此过程作的计划并不是很详细，并不能保证一定能够成功。

事实上，这一过程很可能会做出错误的决定，这是非常有可能发生的事情。

案例

1995 年我刚进微软公司时，当时有人就提出一个新项目，认为微软应该做 Internet 的产品。但是比尔·盖茨却在 COMDEX 会议（计算机分销商展览会议）上说：“我们不做 Internet 的产品。”因此该提议并没有得到通过。但是，很快美国在线公司（AOL）和美国网景公司（Netscape，以开发 Internet 浏览器闻名）就发展起来了，根本原因就是因为他们 Internet 产品非常有市场。

那时我就听到周围的同事说：“Bill made a wrong decision!（比尔做了一个错误的决定！）。”很多人都认为 Internet 将是微软的方向，将是计算机的未来。当时许多人都这么说时，我还觉得很奇怪：他们怎么能够批评老总？后来，很多人就直接给比尔·盖茨发送信件，当然，公司也有许多

人为比尔·盖茨辩护，帮他说话：“我们针对的是桌面计算机，因此做的软件就应该与它有关，为什么还要去做 Internet 呢？”比尔·盖茨收到这么多员工的反馈，于是他自己也怀疑自己是否做错了。因此，比尔·盖茨赶快又派人进行调查，又将这一阶段重复了一遍。结果这一阶段又拖了几个月。其实，在做一项新产品时花几个月的时间进行调查是很平常的事情，因为在进行市场和技术分析的时候，需要收集和查阅很多资料。可是几个月后，结果不需要调查都已经很清楚了。为什么？Netscape 起来了！AOL 起来了！大家都看到了 Internet 这个大趋势，所以，比尔·盖茨赶快调整策略，宣布微软也要做 Internet。

所以，在软件开发的第一阶段，很有可能会做出错误的决定。但是，我知道自从那个错误之后，比尔·盖茨很少再犯这种大错误。因为他从中受到启发：他一个人的智慧不可能盖过所有人的。所以，后来比尔·盖茨非常尊重员工的意见。比如，每年他都要进行几次“思考周（Thinking Week）”，也就是离开公司，到一个安静的地方去考虑一些有关公司的长远发展问题。这些问题从哪儿来的呢？就是全公司的员工提出来的。公司会给所有的员工发一封信，说：“比尔·盖茨有一个思考周，你们可不可以向比尔·盖茨提出一些你们认为对公司的未来有影响、很重要的问题？不管是关于业界的，还是公司的，或者其他方面的问题都可以”。于是，一个个底层小组得到很多意见，然后，公司经过一层层的筛选，将认为很重要的意见提取出来交到上一层小组，后者又将得到的所有意见进行筛选，将认为最重要的问题交给比尔·盖茨。所以，比尔·盖茨出去进行“思考周”的时候，通常会带着很多问题和建议出去。所以，微软公司现在很少再犯上面那种错误了。



我之所以引用上面这个例子，就是想强调：在软件开发的第一个过程中，不可能保证永远正确。连比尔·盖茨都会做出错误的决定，何况一般的人呢？重要的是在发现有错误时能尽快调整，并吸取教训防止未来再犯类似的错误。

因此，在软件开发的第一个过程中，往往因为信息不够，却又匆忙做决定，结果出现很多问题。但是，在这一过程中，每一个团队的作用是什么呢？请见表 4.4 的内容。

表 4.4 软件开发第一阶段各团队分担的任务

团队名称	任 务
产品管理	分析是否应该做这个新产品，尽量证明该产品是值得做的
项目管理	根据产品管理团队决定要做的新产品，设计新产品的目标，以及具体的实现方法
软件开发	开发一些技术原型，来检验该新产品是否有意义，并向人家展示一下新产品未来的样子。另外，还要对开发过程中一些大的结论提出建议
软件测试	判断该新产品是否有用，是否可接受
用户培训	分析用户的需求
后勤管理	对长期的支持管理提出建议

其实，在软件开发的第一阶段，主要任务是产品管理团队（Product management）承担的，其他团队都是做些辅助工作。

当软件开发第一阶段结束时，我们应该获得哪些可交付产品呢？主要是有关新产品想法和意图的文档。如表 4.5 所示。

表 4.5 软件开发第一阶段可交付产品

可交付产品	发布类型	变 更 控 制
想法和意图文档	对外	基线（Baselined） 由产品管理团队控制
风险管理计划	内部使用	是动态变化的。由开发团队的头控制
产品结构	对外	是稳定的。由项目管理团队控制

其中，风险管理计划是为了保证我们的开发进程。另外，产品结构不一定要要求很详细，只需要一个大概的描述即可。

在这一过程，有许多任务（Task）需要处理。这些任务只有完成

了，才能促使我们关于新产品的想法和意图得到批准。这些任务如图 4.9 所示。

从图 4.9 中可以看出，首先我们要根据需找到一批团队的关键和主要成员。我以前的一个老板就是这样做的，每次开始一个新项目时，他首先找几个主要的人员，包括开发人员和项目经理。

接着，开始定义新产品的想法和意图（Define Vision and Scope）。例如，如果想做 XML，就需要对 XML 进行定义，并详细论述 XML 的好处和作用。

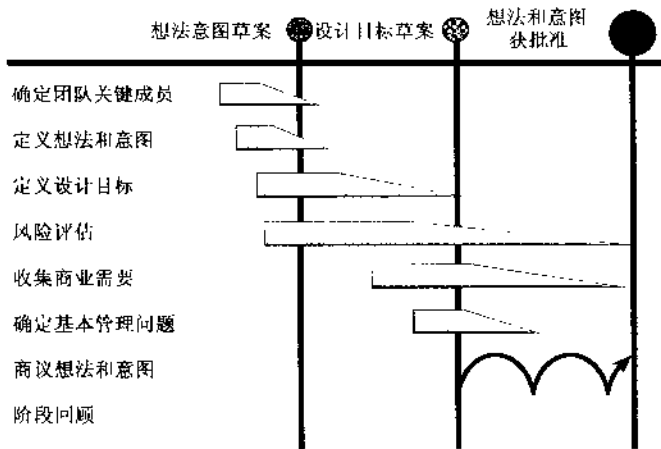



图 4.9 软件开发第一阶段需要完成的任务

第三个任务是定义设计目标（Define Design Goals）。例如，设计 XML 的目标就是希望 XML 能够为用户提供一种在 Web 上创建信息格式并共享格式和数据的灵活途径，从而摆脱目前存在的问题。我与我在 SQL Server 团队时的老板聊天的时候，他就说：“我的目标就是让所有的产品都能用上 XML。今后，只要你能上网，就能使用 XML。”微软应用 XML 技术的第一产品就是 SQL Server。当时我所在的团队就是 SQL Server XML，它的目标是：只要有 Internet 的地方，就能用 SQL Server 数据库通过 Internet 进行管理。当时，我知道了这一目标后，觉得这真是一个非常令人兴奋的项目。



第四个任务是风险评估 (Risk Assessment)，要分析清楚各种风险。

第五个任务当然是要收集商业需求 (Gather Business Requirements)。例如，我做 Windows CE 的时候需要什么？为什么要做这个新产品？要将这些要求收集起来，经过整理，然后清楚地写出来。

第六个任务是确定基本管理问题 (Identify Infrastructure Management Issues)。要找出在这个过程中遇到的一些问题，例如有些管理方面的问题，是不是在这个过程中一定会得到支持，会坚持下去，等等诸如此类的问题。

第七个任务是商议新产品的想法和意图 (Negotiate Vision and Scope)。从图 4.5 中我们可以看出，这个任务是需要花费一定时间的，因为这是一个很大的目标。例如，有些人说：“我希望该新产品做出来后，可以让计算机能听、能说。”可有些人却说：“我希望该新产品可以让计算机能听就行了。”可是还有一些人希望计算机能够将 Internet 上的内容“说”给他们听，从而他们就能在家里闭上眼睛，更轻松“浏览”Internet 的内容，所以他们会说：“我不在乎，我只希望它能说就行了。”所以，各人的目标不一样，必须平衡一下，得到新产品的最终想法和意图。

第八个任务是阶段回顾 (Milestone review)。

以上就是微软软件开发的第一个里程碑。

4.5 产品计划的通过里程碑

计划过程微软软件开发的第二个阶段，主要是项目经理做的工作。要想通过项目的计划，必须将项目的所有计划具体化定义出来。

在这一阶段，我们需要在以下几个方面达成一致。

■ 项目陈述 (Project Deliverables)

主要陈述该项目可以为公司带来的好处，可以拿得出手的东西。

■ 功能和优先级 (Features and Priorities)

详细定义产品的功能，以及它们的优先级。例如，前面提到的“语音识别”项目，目标是让计算机既能“听”又能“说”。那么，这两个功能谁更优先呢？一般来说，人们可能认为“能说”比“能听”更容易实现，所以我们先满足“能说”这一功能。

■ 项目风险 (Risks)

详细分析我们在做这个新项目的过程中会遇到的所有风险，并判断我们能否控制这个项目的进展。例如，我们是否能够按时完成？我们的技术是否已经相当成熟？还是拿“语音识别”这个例子来说，我们的识别率是否成熟到让人们接受的程度？如果人们不满意，我们这个产品可能就失去意义了。

■ 发布日期 (Ship date)

对产品的发布日期，大家都要达成一致意见，要保证到时能够完成，并让用户使用。

■ 商业要求 (Business requirements)

从商业的角度对产品提出各种要求。

■ 产品功能规范 (Functional Specification)

详细定义产品的各种功能，将产品的各种功能具体化，并使大家达成一致看法。

■ 产品计划 (Project plan)

产品总体计划，包括人员分配和结构组成。



■ 产品进度表（Project schedule）

这是产品的总体进度表。

在这一阶段中，每一个团队的作用是什么呢？请见表 4.6 的内容。

表 4.6 软件开发第二阶段各团队分担的任务

团队名称	任 务
产品管理	概念设计和市场推销计划/进度表
项目管理	逻辑设计、功能规范，以及总体计划/进度表
软件开发	物理设计和开发计划/进度表
用户培训	用户性能支持设计和用户培训计划/进度表
软件测试	设计评估和测试计划/进度表
后勤管理	设计评估和后勤计划/进度表

从表 4.6 中我们可以看出，在软件开发的第二个阶段，产品管理人员需要将第一阶段抽象定义的想法和意图具体化，变成概念性的东西，并开始做一些市场推销工作。事实上，很多产品还未做出来，但我们都已经知道了该产品，这就是产品经理们做的工作，他们让人们很早就知道了将有这么一个新产品。

当然，软件开发的第二阶段主要是项目经理的事情。他们首先需要进行产品的逻辑设计，然后定义产品的功能规范，再制定较高层次的项目总体计划和进度表。只有知道了产品各功能的细节，才能估计产品开发需要的时间、人力，以及开发方法和过程。

在此阶段，软件开发人员开始进行物理设计、设计代码，并且需要制定出自己的开发计划和进度表。

用户培训人员主要是从用户的角度出发，提出新产品应该具有的功能，帮助项目管理人员进行产品的设计，并制定用户培训计划和进度表。

软件测试人员评估产品设计是否正确，例如，产品的目标是实现计算机“能听”、“能学习”，但是，如果目前的技术还不成熟，那

么这个设计是有问题的。同时，测试人员还需要制定自己的测试计划和进度表。

最后，后勤人员做一些产品开发的准备工作，例如：提供设备等。

当软件开发第二阶段结束时，我们应该获得哪些可交付产品呢？如表 4.7 所示。

表 4.7 软件开发第二阶段可交付产品

可交付产品	发布类型	变更控制
功能规范	对外	由产品管理团队控制
风险管理计划	内部使用	是动态变化的，由开发团队的经理来控制
项目总体计划	对外	是稳定的，由项目管理团队控制
项目总体进度	对外	是稳定的，由项目管理团队控制

综上所述，软件开发的第二阶段主要经过如图 4.10 所示的几个过程。

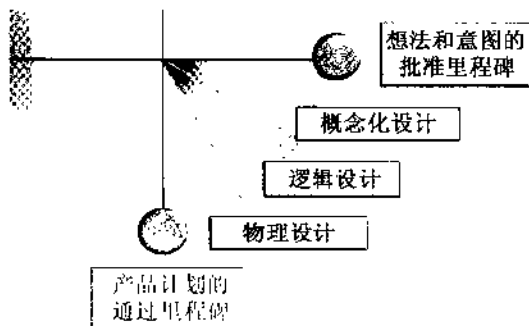


图 4.10 软件开发第二阶段经过的过程

首先，当新产品的想法和意图提出来后，我们需要将这些最初设计的大概的思想转化成一种具体的、概念化（Conceptual）的东西，接着再将它转化为符合逻辑（Logical）的设计，然后再到具体的、基本的物理（Physical）设计。只有经历了以上过程，才完成了软件开发的第二阶段，产品计划才被批准。



在软件开发的第二个阶段，不同角色的人员对新产品的判断标准是不同的。一个产品规范只要能够让不同的人看了以后都可以找到自己所需要的东西，就是一个好的产品规范。例如，表 4.8 中的内容就显示了一个好的产品规范能够让不同角色的人阅读后找到他所需要的东西。

表 4.8 一个好的产品规范的判断标准

角 色	标 准
用户	系统能给他带来商业利益
产品管理人员	系统满足已知的要求
项目管理人员	职责和提交的进度表是稳定的、可行的
开发人员	能够根据设计编程，并且实现的风险是可接受的
测试人员	测试策略和规范所需的平台、脚本和数据是完整的
用户培训人员	用户的需求已定义
后勤人员	所有组织、应用及系统接口都是确定的
所有人	新产品发布日期确定得比较合理

在这一过程中，有许多任务（Task）需要处理，如图 4.11 所示。

从图 4.11 中我们可以看出，软件开发的第二个阶段有许多任务，开始阶段包括：概念设计（Conceptual Design）、逻辑设计（Logical Design）和物理设计（Physical Design）。其中，物理设计就是转变为易于编程的功能设计。这三个过程是以一种循环方式进行的，并且是从概念设计开始的，例如，开始定义的概念为“语音识别”。但是，做了一段时间之后，到了物理设计，发现开始定义的概念不太准确，应该修改为：“某种条件下的语音识别”或“某个范围内的语音识别”。这样，又需要回到概念设计过程进行修改。因此，这一过程是根据情况而变化的，可能需要反复进行。

一旦有了物理设计，我们就需要进行风险评估（Risk Assessment），分析开发新产品存在的各种风险。并且还要进行功能规范的商议（Functional Specification Negotiation），对一些事情进行取舍。例如，一般来说，如果要求新产品开发的周期越短，那么它具备的功能就越少。所以，大家需要一起协商，然后做出决定。

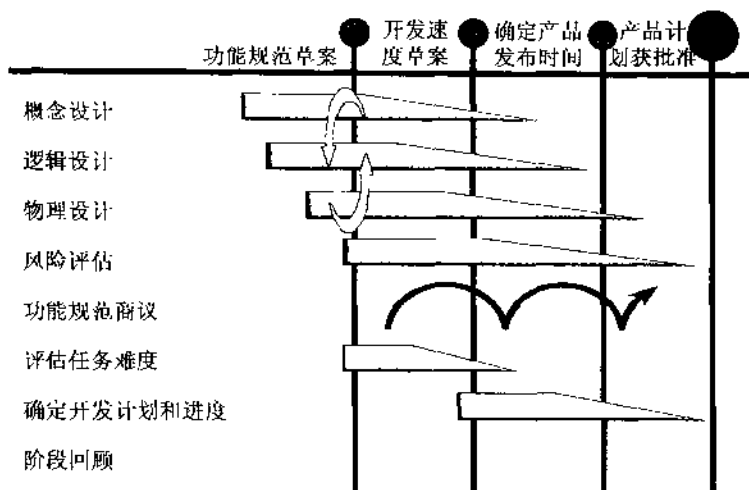


图 4.11 软件开发第二阶段需要完成的任务

接着，还需要估算新产品开发的难度（Task-Level Estimating），如技术上要解决的问题、需要的人力、财力和时间等。

然后，做出具体的产品开发计划和进度表（Planning and Scheduling）。

最后，进行阶段回顾（Milestone Review）。

4.6 范围完成/第一次使用里程碑

开发过程是微软软件开发的第三个阶段，主要是开发人员做具体的编程工作，同时，测试人员开始进行软件测试工作。在微软，只要开发人员一开始编程，测试人员就要开始进行测试工作。

在这一阶段，我们需要在以下几个方面达成一致。

◀ 功能规范化（All features built to specila）

对所有功能的描述都要更加规范。所有功能都需要根据项目经理制定的功能规范来编程。





- 、使用 and 估计产品 (Using and evaluating the product, even if testing is not complete)

使用和评估这个产品，这只是开发人员自己进行的一些测试。此时，测试人员可能还未正式开始测试。

- 、用户试用 (Customers using the product for the first time under beta conditions)

在产品代码的编写快完成时，用户就开始试用。当然，公司内部是最先试用的。

- 、基础要求的配置 (Deployment of infrastructure required for first use)

第一次使用的时候，要保证所有的基础要求都得到正确配置。

- 、配置计划细节 (Deployment plan details)

微软开发产品的过程就像是建房子的过程。首先要搭架子，然后再筑墙。但是，也有其他一些公司在做产品时，是先把产品各个部分都开发好了，再将它们连接起来。这就相当于先把所有的房间都修好了，然后再将它们合并成一个房子。其实，这样的做法是很别扭的，通常很难获得好的成果。因为你已经把它们（房间）固定了，再合成的话非常难。而在我们开发的过程中，每天都有一个软件构建（build）过程。也就是说，每天每一个开发人员所写的程序都需要进行提交（Check-in）操作，我们有一个专门的软件构建小组，他们就负责将这些提交的程序合并起来，并编译链接成一个产品版本。尽管在软件编码的开始阶段，功能还很不全，只是很少的一部分，但是有多少，就连接多少。于是，开发人员之间可以互相使用所有已有的代码。微软之所以总是能够很好地控制由几千个工程师组成的团队共同参与开发一个项目，最后完成非常大的产品，原因就在于：一开始就将各开发人员的代码连接在一起，并在开发的过程中不断地补充新的程序。

我问过其他一些开发大型软件的公司人员，他们都说大型软件做到最后非常困难。因为他们首先将每一个特性的实现程序都同时

发展起来，但是从开始就没有将它们连接在一起，直到最后才连接它们，这时的工作就非常困难了。例如，每个房间都做得很漂亮，单独看都非常好，但是要将它们合成一座房子就可能会出现問題，因为有可能这个房间高一点，那个却又低了一点。

在软件开发的第三阶段，每一个团队的作用是什么呢？请见表 4.9 的内容。

从表 4.9 中我们可以看出，在软件开发的第三个阶段，产品管理人员需要控制用户的期望值。因为这时程序快完成了，所以产品到底具有哪些功能，产品管理人员心里是非常有底的，他们可以向用户透露一些信息。当然，这时产品管理人员也开始推销自己的产品，估计产品的价格，以及包装的内容。

表 4.9 软件开发第三阶段各团队分担的任务

团队名称	任 务
产品管理	控制用户的期望，推销，价格，包装
项目管理	项目跟踪/沟通，Beta 版本计划
软件开发	特性开发/测试
用户培训	用户性能支持开发
软件测试	测试，保证稳定性
后勤管理	渠道沟通和首次展示计划

在第三阶段，项目管理人员一直都很忙，需要进行产品跟踪，保证产品按时完成，并准备产品的 Beta 版本计划。所以，项目经理每天都要与开发团队中其他人进行沟通。记得当时我们开发产品时，项目经理一会儿要问测试人员：“今天有多少 Bug？”一会儿又会问开发人员：“今天程序还稳定吗？通过基本测试没有？”每天，项目经理都要到处检查，非常辛苦。

软件开发的第三个阶段主要是软件开发人员的工作，即进行程序开发，并且开发人员还需要自己进行测试。

用户培训人员开始准备用户需要的东西。首先他们需要了解该产品，然后准备去培训用户。

软件测试人员需要制定一套测试标准，保证产品合格，并要写出完整的测试计划和测试方案。

案例

我为大家举个例子，说明如何判断产品是否合格的问题。例如，如何判断程序的性能优劣？其实这很简单，让现在已有的产品和新产品在同一台机器上运行，例如访问数据库，并返回结果。然后，我们查看两种产品返回的时间就可以了。根据结果，就可以判断性能的提高率。又如，对于 E-mail 系统，看看十万个用户同时使用的时候，该系统会不会崩溃？会不会出现问题？如果一切正常，可以说此 E-mail 系统性能很好。根据我们的经验以及研究调查，在实际中，十万个用户同时使用 E-mail 系统的可能性很小，即使一个有几万人的公司（如波音公司），所有员工同时访问服务器收发 E-mail 的可能性也很小。所以，只要十万个用户的测试通过了，我们就可以说该 E-mail 系统的强力测试（Stress test）成功通过。

在这个阶段，后勤管理人员做些其他的工作，例如，制定产品首次展示的计划。

当软件开发的第三阶段结束时，我们应该获得可交付产品？如表 4.10 所示。

在第三个阶段结束时，代码基本上已经完成，所以我们可以拿到一个完整的程序，也有了产品的最初模型。这样，产品的功能规范就基本固定了，因为代码都编写出来了，不需要再进行很大修改。其实，在软件开发的过程中，只要代码还没有编写完成，产品的功能就一直在不断修改。

表 4.10 软件开发第三阶段可交付产品

可交付产品	发布类型	变化及控制
功能规范	对外	内容固定了
风险管理计划	内部使用	是动态变化的，由开发团队的经理控制
源代码和执行程序	内部使用	由开发人员控制
性能支持元素	对外	由用户培训团队控制
测试规范和测试案例	内部使用	由测试人员控制
总体产品计划和总体产品进度表	对外	稳定的，由项目管理者控制

案例分析

我记得最初我并不知道功能设计也是可以修改的，有一次，我发现产品的设计实在不合理，我与开发人员聊天的时候，就说了我的想法，开发人员就说：“那你为什么不将它定为一个 Bug？”我觉得很奇怪，就问：“还能说功能设计有 Bug？”他说：“只要你有道理，当然可以。”不过，功能的更改非常不容易，所以，我在报告功能设计的 Bug 时都比较小心。如果我认为某功能设计不合理，我就要提出一个更合理的设计，并且，我会去找开发人员：“你觉得我这个设计是否能够实现？”开发人员就会帮我分析：这个方案对开发的影响大不大？会不会引起别的问题，等等。如果开发人员认为这个方案是可行的，付出的代价不大，我才真正报告功能设计的 Bug。所以，往往项目经理问：“你为什么说我的设计有问题？”我会将自己的想法告诉他，并将开发人员找来一起讨论。项目经理经过仔细考虑以后，觉得我提出的设计有道理，并且开发人员也说没有问题了，项目经理就会同意对设计进行修改，因为这样能使产品更好，他何乐而不为呢？我曾经找出过 40 个功能设

计 Bug，其中有 90% 的 Bug 都被承认了，并进行了更正。之所以采用率这么高，主要原因就是我已经先与开发人员讨论过了，开发人员已经说可以实现，所以项目经理就很容易接受新建议了。而且，这样一来，我的信誉也就越来越高，他们也就越来越容易接受我的建议了。但是，一旦程序编写完毕，就再也不能更改功能设计了。

在这个阶段结束时，还能得到风险管理计划文档，就是讲述如何控制风险文档。例如，某一部分可能有风险，有时开发人员自己知道，他就会告诉项目经理：“这一部分可能不稳定，因为代码太多……”。对自己的程序，编程人员通常都有感觉：哪些部分很有把握，哪些部分没有把握。其实有时候很有把握的部分也会有问题，只不过自己不知道而已。

另外，还能得到源代码和可以运行的程序，这是由开发人员控制的。同时，测试人员的测试规范和测试案例也全部写出来了。最后，因为代码已经都编写完毕，总体产品计划和总体产品进度表也就可以更新了。这时候，我们就可以向外界宣布该产品可以发布的大概日期。但是没有办法准确地预测产品的发布日期。因为影响一个产品的因素很多，如工程师的水平、开发阶段、开发人员的心情等。这些因素在实际中都会影响产品的开发，而且许多因素还没有人能够说得清楚，例如，如果当时开发人员的精力很充沛，状态很好，那么产品质量就会很高；可是，如果他们太累了，那么犯的错误就会很多。这些因素都会影响产品的发布日期。

在这一过程，有许多任务需要处理，如图 4.12 所示。

为了与其他产品兼容，我们不仅要开发产品本身，还要做许多其他的开发，包括用户界面开发（User Interface Dev）、用户服

务开发 (User Services Dev)、商业服务开发 (Business Services Dev)、数据服务开发 (Data Services Dev)、数据库开发 (Database Dev) 和内容开发 (Content Dev) 等。

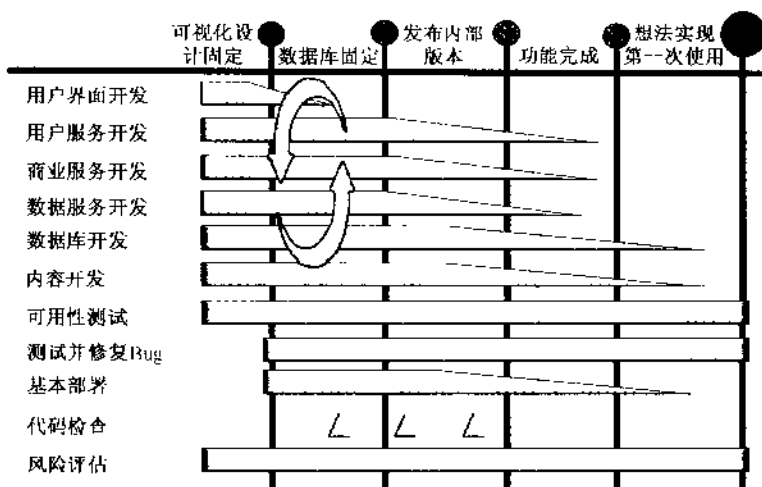


图 4.12 软件开发第三阶段需要完成的任务

另外的一个任务就是可用性测试 (Usability testing)。有一次，我买了一个软件，安装之后，我一直没有关机。过了几天，我发现我的机器越来越慢，到后来就死机了。我一看就知道，这个软件有内存泄漏问题，它不断“吃”机器的内存，直到最后机器没有内存了，就会死机。在产品测试工作中，这种事情是很常见的。只要我发现机器越来越慢，就知道一定有内存泄漏。这时只要用我们编写的工具来测试，最后一定能找到问题的所在。这种内存泄漏速度很慢，有时候，一天仅仅“吃”掉几 KB 内存。但是，只要内存泄漏问题存在，该产品就是不行的。但是，有些产品根本没有这些测试，兼容性也有问题，经常会和另外一个已经安装的产品发生冲突。

从图 4.12 中我们可以看出，只要开发人员开始编写程序，那么测试和更正 Bug 的任务 (Testing and bug fixing) 就开始了。

在软件开发的第三阶段，风险评估 (Risk assessment) 也是一直



要做的一个任务。

在这里，我有一点声明：图 4.12 中表示的各个任务的时间不一定很准确，不同的产品会有所不同，这里只是为了给大家一个概念。例如，Exchange Server 好几年才发布一个新产品，Windows 操作系统一般也是两到三年发布一个新产品，它们的任务开发过程就可以长一些，但 Internal Explorer 却是一年一个版本，它的任务开发时间就短一些。所以，根据产品的功能不同，会有不同的任务时间表。

4.7 发布阶段

我们已经知道了，在微软的软件开发四个阶段中，第一个阶段主要是产品管理人员的工作，第二个阶段主要是项目管理人员的工作，第三个阶段主要是开发人员和测试人员的工作，那么最后一个阶段我们主要做些什么工作呢？可以这么说，这一阶段主要还是测试人员的工作，目的就是为了保证产品的质量。发布阶段的过程如图 4.13 所示。

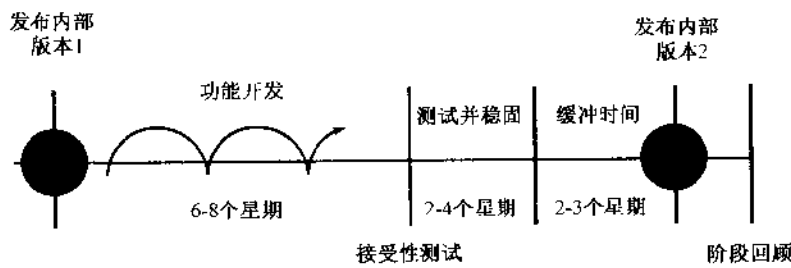


图 4.13 软件开发的发布阶段过程

从图 4.13 中可以看出，软件开发的发布过程可以有 6 到 8 个星期的功能开发时期，2 到 4 个星期的测试和稳固时期。一般来说，微

软每三个月就会设定一个内部的里程碑（Milestone），我们称之为 Alpha 1 版本、Alpha 2 版本……。不同产品的 Milestone 时间是不一样的，如有些产品是半年为一个里程碑。图 4.13 中所示的“缓冲时间”阶段，就是在不做太多修改操作的条件下，测试软件的运行情况，直到产品的最后完成。

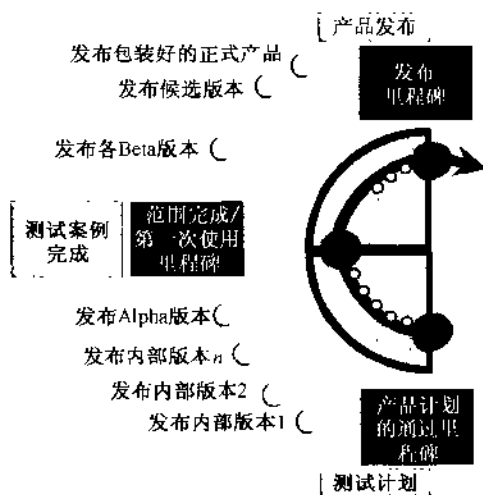


图 4.14 完整的测试过程

从图 4.14 中我们可以看出，从软件开发的第二个阶段，直到产品的最后完成，最忙的事情是什么？就是测试！恐怕这一点大家都没有想到。我们可以回忆一下软件开发的四个过程，从第二个过程开始，测试人员就需要帮助项目管理人员评估产品的设计。当项目经理开始编写产品的功能规范时，测试人员就需要开始编写测试规范。我经常就是在项目经理的功能规范还没有写完时，开始写我的测试规范，所以当他写完了功能规范后，我基本上也完成了自己的测试规范。然后，一旦开发人员开始编写代码，测试人员就需要开始进行测试工作，并且还需要编写所有的测试文档并设计测试案例。所以，在这个阶段，开发人员忙于写代码，而测试人员同样也很忙，因为他需要进行各种测试。到了第四个阶段，完全就是测试人员的



工作，因为这时开发人员的代码已经编写完毕，他们的工作就是更正 Bug，而这依赖于测试人员的测试结果。所以，在软件开发的第四个阶段，测试人员的任务非常重。

现在大家都很重视编程人员的培训，但是不太重视测试。其实测试是非常重要的。在微软，测试人员是最多的，如果产品团队由十个人组成的话，测试人员要占一半以上。

编者注：为了让大家能够更加了解测试的过程及方法，陈宏刚博士还将自己在微软从事软件测试的多年经验总结成文，参见本书第 9 章、第 10 章。

从图 4.14 中还可以看出，软件开发过程中的测试工作需要经历：测试计划（Test Plan）、发布内部版本 1（Internal Release 1）、发布内部版本 2（Internal Release 2）、发布内部版本 n （Internal Release n ）、发布 Alpha 版本、发布各 Beta 版本、发布候选版本（Release Candidates）、发布包装好的正式产品（Golden Release）这几个阶段。一般来说，当项目管理人员的产品功能规范写完之后，测试人员的测试计划也就应该编写出来，其中包括测试案例。但是，直到开发人员的代码编制完毕，即 Alpha 阶段，测试人员的测试案例才真正结束。因为在这一过程中，测试人员的测试案例是需要不断更新的。例如，如果在开发过程中，项目管理人员制定的产品特性改变了，那么测试人员的测试案例就必须相应地做出变化，同时，测试规范也要进行更改。另外，如果产品增加了新的特性，那么就需要增加许多新的测试案例；如果删除了已有的特性，那么测试人员也需要去掉相应的已有测试案例。当开发人员的代码全部完成后，也就是说，产品的特性不会再更改，这时，测试案例也就确定了。

当测试案例确定后，就有一部分测试人员根据这些案例开始编写测试工具，专门用于产品的自动测试，而且这些工具能够自动判断产品的错误。一般来说，测试工具的运行范围很广，它能够处理很多复杂的情况。有些情况利用人工很难处理。





为什么需要测试工具呢？

例如，需要五万人同时发送 E-mail 的情况，这在现实生活中很难人为实现。但是，利用测试工具就可以非常容易地做到。测试工具可以自动产生 50 000 个账号，并且让它们在同一时间从不同机器上（一个机器上可以有多个不同的账号）同时发送 E-mail 信息。再举一个浏览 Web 页面的例子，如果要求用户连续浏览 24 小时几万 Web 页面，一般来说，用户是不可能连续不断浏览这么多的页面的，但是机器是可以做到的，并且工具还可以自动判断浏览结果是否正确。

在软件开发的第四阶段，具体地说，我们需要在以下几个方面达成一致。

◀ 产品稳定和所有问题的解决（Product stability and resolution of all issues）

找出所有的问题，并更正那些影响使用的 Bug。我必须告诉大家一点：“所有的产品都有 Bug！”不论一个多么小的产品，我还是相信它存在 Bug，只不过这些 Bug 不影响你的使用而已。例如，显示图形时如果只有几个像素的显示位置出现错误，人的眼睛是看不出来的，但是机器是知道的。这就是一个 Bug，只不过这个 Bug 可以不用更正。

◀ 用户接受产品（Customer acceptance of the product）

◀ 转移长期管理和支持的所有权（Transfer of ownership for long-term management and support）

◀ 关注下一版本（Change in team focus to the next release）

在这一阶段中，每一个团队的作用是什么呢？请见表 4.11 所示。





表 4.11 软件开发第四阶段各团队分担的任务

团队名称	任 务
产品管理	Beta 版本位置管理、推销和开始准备产品发布
项目管理	Beta 版本管理、项目跟踪/沟通、控制产品开发活动的结束点
软件开发	更正 Bug、产品发布
用户培训	最终产品、培训教员、发布
软件测试	测试和 Bug 报告
后勤管理	建立 Beta 版本位置并提供支持、产品首次展示的管理

在软件开发的最后一个阶段，因为代码已经编写完毕，产品管理人员就开始准备产品发布的一些工作，例如协调 Beta 版本的发布情况。Alpha 版本是微软内部人员使用的，但从 Beta 版本开始就需要让外界的用户使用该产品了。微软有许多的试用用户，例如：福特公司的几十万员工、波音公司的几十万员工等。现在，国内也有越来越多的用户参加我们的试用活动。例如，Visual Studio .NET 产品的 Beta 2 版就给了北京大学和清华大学各 1000 份，供大家进行测试。

我们的项目管理人员需要管理各个 Beta 版本，跟踪产品的开发过程，保证产品按时完成，并且控制产品开发活动的结束点（Sign off）。其实，在这个过程中，有些项目管理人员已经开始准备下一个版本，开始编写下一个版本的功能规范了。

开发人员主要做的工作就是更正 Bug，并进行下一版本研发的准备工作。

用户培训人员要准备最终产品并培训教员。

前面我们已经说过了，软件开发的最后一个阶段主要是测试人员的工作，这时，他们需要进行软件测试并提交 Bug 报告

后勤管理人员主要是建立并支持 Beta 版本放置的站点，并负责产品首次展示的管理。

当软件开发最后一个阶段结束时，我们应该获得哪些可交付产品呢？如表 4.12 所示。

表 4.12 软件开发第四阶段可交付产品

可交付产品	发布类型	变化及控制
包装好的产品	对外	冻结
发布备忘录	对外	冻结
性能支持元素	对外	由操作或支持控制
测试结果和测试工具	内部使用	存档
源代码和可执行程序	内部使用	存档
项目文档	内部使用	存档
阶段回顾	其他项目团队	稳定的由项目管理人员控制

在软件开发的第四个阶段结束后，我们能够得到的东西首先当然就是已经包装好的最终产品，还有发布备忘录。这时，它们都被冻结了，不会再有变化。

在软件开发的过程中，我们不能保证已有的东西都不变，所以，我们的操作手册和帮助文档都是到了 Beta 阶段才开始写。

在软件开发过程结束时，我们必须具备所有的性能支持元素。还有许多内部使用的测试结果和测试工具，以及一些还未解决的 Bugs，希望在下一个版本中更正它们。另外，还有项目管理人员、开发人员及测试人员编写的所有项目文档。

在这一过程，有许多任务（Task）需要处理，如图 4.15 所示。

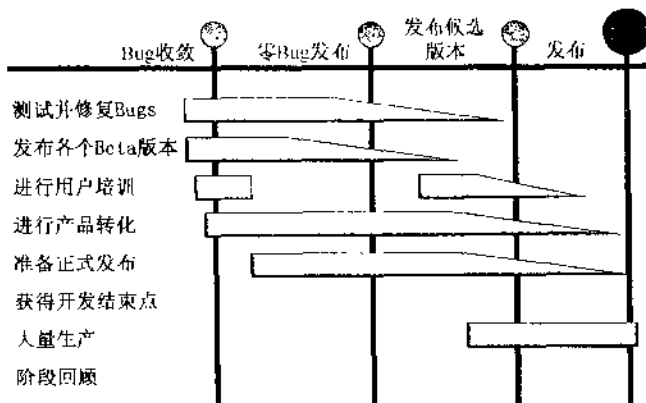


图 4.15 软件开发第四阶段需要完成的任务



软件开发的第四个阶段有许多的任务。图 4.15 中有一个“零 Bug 发布 (Zero Bug Release)”过程，这并不是说没有 Bug，而是说将所有的 Bug 按优先级排序，如果认为必须更正的，就将它记录下来，称之为“活动的 Bug (Active Bug)”；否则就将它搁置在一边，不用更正，或者推迟到下一个版本来更正。例如，有一些 Bug，它们并不影响使用，我们就可以认为它们不需要更正。如果将后一种 Bug 都排除之后，剩下的“Active Bug”数目为零，我们就可以说“Zero Bug”。

从图 4.15 中可以看出，测试和修复 Bug 这一任务从软件开发的这个阶段开始就一直在做。

另外，从前面的分析可以看出，微软在开发产品时严格遵守了“基于版本发布”的指导原则，这样做的好处有：

- ◀ 在每次发布一个版本时都会促使产品中的问题被终止或解决
- ◀ 使所有的团队成员都有一个清晰、明确的动力和目标
- ◀ 可以在产品范围之内对不确定性因素进行管理，并适当进行修改
- ◀ 可以保证每个发布版本的连续性，并且功能越来越多，越来越强
- ◀ 可以保证发布到市场所需的时间更短

图 4.16 中显示了“基于版本发布”的特点。可以看出，随着时间的推移，发布的版本越来越新，功能也越来越多，越来越强。

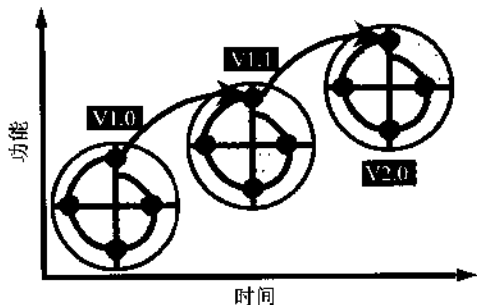


图 4.16 “基于版本发布”的特点

在这一过程,我还想给大家重点介绍一个概念: Trade-off Decision (折中决定)。其实,大家都知道,很多时候,我们一方面希望我们的产品质量很可靠,但另一方面又希望产品越早发布越好,因为我们做产品就是希望能够赚钱,这是软件开发商的最终目的,同时我们还希望产品的功能越多越好。可是这几样东西是互相矛盾的。如果要是保证质量,那么产品的发布时间就可能要拖后,或者产品的功能就要减少;如果产品的发布时间要加快,那么产品的质量就可能得不到保证,或者产品的功能就要减少;如果产品的功能越多,那么产品的质量可能就不可靠,或者就不能按时完成。那么到底应该怎么做呢?我们有一个建议,即“Trade-off”建议:产品的可靠性能够介于最优和可接受之间就可以了,产品的发布日期当然是越快越好,产品能够实现预期的必须功能就行了。通常,在一个最恰当的时间发布一个最好的产品需要做一些非常巧妙的权衡。只要你为一个产品做了正确的 Trade-off 决定,用户就会相信该产品是高质量的。图 4.17 所示的就是我们认为是一个正确的 Trade-off 决定。

	最优	约束范围	可接受
可靠性		✓	
发布日期	✓		
功能			✓

图 4.17 正确的 Trade-off 决定

微软最重要的一个开发策略就是:对产品的开发做一个正确的 Trade-off 决定。

当所有的过程都完成以后,产品就可以发布了。此时我们已经完成了下述工作:

◀ 已经做了团队所有的决定





- ◀ 测试工作已经完成
- ◀ 已经做了 Triage 决定

在产品到了 Beta 版本阶段以后，会由项目管理团队、开发团队和测试团队各出一个领导，形成一个新的团队，我们称之为“Triage”。“Tri”的意思就是三方面的。这三个人需要在一起共同决定需要更正的 Bugs，以及可以搁置的 Bugs。越到产品开发的后期，更正一个 Bug 产生的影响就越大。因此做出决定是非常困难的，常常要吵得不可开交。项目管理人员希望尽早发布产品；测试人员认为应该保证质量，保证用户的使用；而开发人员觉得多一事不如少一事。

■ 产品终结

在微软，每一个团队都要签字以终结产品的开发，包括项目管理团队、开发团队和测试团队。

案例分析

我记得在 IE Beta 2 版本阶段，就在要签字的时候，我又发现了一个 Bug，并且认为该 Bug 很重要，但项目管理人员说这个 Bug 完全可以搁置不管。我认为这个 Bug 将影响用户的使用，所以我坚决拒绝签字。这样，该产品就不能发布，这是我们公司的规矩，只要有一个团队拒绝签字就不能发布该产品。所以，后来我们产品的总经理都来找我了，问：“乔治，你为什么不签字？”我就向他说出我的看法。他说：“我看了你的报告，但是你使 Bug 重现的那些步骤很奇怪，哪有用户会像你这样来回折腾，做那么复杂的操作？”我说：“好，我会尽快给你一个简单的操作步骤。”我马上把我的团队召集起来，一起攻关，后来我们找到两种办法，让该 Bug 快速就重现了。实际上，我事先已经找了开发人员，他们已经帮我

找到了出现该 Bug 的原因。既然知道了原因，当然很快就能找到使该 Bug 快速重现的测试案例了。最后，总经理看到该 Bug 这么简单就出现了，认为产品确实有问题。于是，公司做出了决定，推迟发行 IE，就是为了更正该 Bug。其实开发人员只需要半个小时就可以更正该 Bug，但是，测试人员却需要花两天时间进行测试。这是为什么呢？因为各个 Bug 是互相牵连的，修改了这个 Bug，可能会引发其他的各种 Bug，因此需要对所有出现过的 Bug 全部重新测试，这个过程是非常艰难的。这也正是为什么在此阶段大家都不愿意更正 Bug 的原因。

当所有的测试都通过之后，就是庆祝和获得赞誉的时刻了！这个产品就完成了！

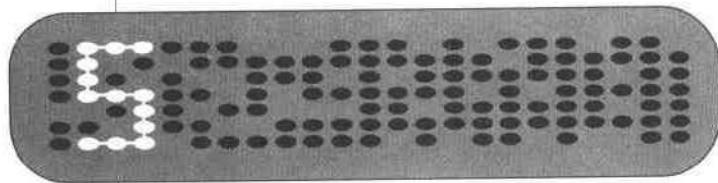
第5章

软件设计之源

Source of the Software Design

背景

在有些人眼里，今天的软件开发似乎已成为简单的事件：已有了不少很好的开发工具和软件库，软件开发人员训练有素，都强烈渴望去编写很酷的软件，可以在几天的时间里编写出一个相当复杂的软件。但为什么有一些软件能够得到用户的喜欢，而另一些则不能？为什么有些软件能够在市场上成功，而有些则受到冷落？由此可见，开发软件并不一定难，难就难在如何开发有用的软件。本章，凌小宁博士就根据自己多年的实践经验，回答“如何设计有用的软件”这个问题。





凌小宇博士在讲解“软件设计之源”

本章内容概览

- └─ 软件设计简述
- └─ 三个困难的问题
- └─ 设计之源
- └─ 错误设计之源
- └─ 基于用户情景的设计

5.1 软件设计简述

本章要回答的问题是：如何设计有用的软件。

在很多人眼里，今天的软件开发似乎已成为简单的事情：已有了不少很好的开发工具和软件库，软件开发人员训练有素，都渴望去编写很酷的软件，甚至可以在几天的时间里编写出一个相当复杂的软件。但事实上，有些软件能够得到用户的喜欢，而另一些则不能；一些软件能够在市场上获得成功，而有些则受到冷落。由此可见，开发软件工作本身的确已变成相对容易的事情，而开发有用的软件才是真正的困难之所在！

一个有用的软件应该能帮助用户解决实际问题，应该能体现出对用户的价值。因此在设计一个软件时，首先要想的应该是：

- ◀ Who（为谁设计，用户是谁）？
- ◀ What（要解决哪些用户问题）？
- ◀ Why（为什么要解决这些用户问题）？

这就是称之为 3W 的软件设计的出发点。如果没有搞清楚这些，即使能很快地开发出自己的软件，也不可能在市场上获得成功。

5.2 三个困难的问题

Who? What? 和 Why? 这三个问题是任何商业软件设计的出发点。它们看起来很简单,但实际操作时却常常发现这是非常困难的问题。只要想一想有成千上万的软件得不到用户的喜爱而在市场上遭到失败,就能看到这个问题的困难程度和重要性。

下面来研究一个实际例了。

案例分析

在 1996 年前后,微软将其掌上电脑操作系统 Win CE (1.0) 投放市场。一些人预测凭借微软强大的市场运作力量定能获得成功。而另一些人则批评说它的用户界面并不适用于掌上电脑用户,更像是台式电脑的用户界面。实际证明那些批评者是对的,Win CE (1.0) 在市场上并不成功,用户不买账,觉得它不好用。Win CE (1.0) 的竞争对手继续控制着掌上电脑市场,而 Win CE (1.0) 渐渐从市场上销声匿迹。

原因何在?

用过 Win CE (1.0) 的人会发现用户的批评是对的。Win CE (1.0) 有太多层次的视窗和菜单。而掌上电脑的用户则喜欢更简洁的界面和更快捷的操作。Win CE (1.0) 的设计人员没有充分理解他们的用户,而是简单地沿用了他们设计台式机界面的经验,因而铸成此错。所幸的是,Win CE 的设计人员及时了解到用户的批评,在以后的版本上做了大量的改进,使得后来的 Win CE (PocketPC) 大受欢迎,很快在企业用户这一市场上超过了竞争对手。



试想，如果微软没有足够的人力、财力来纠正 Win CE（1.0）的不足，Win CE 早已在市场上消失了。试想，如果一个小公司犯了这样一个错误而又没有财力来纠正，这家公司会面临怎样的命运呢！

由此可见，3W 这一软件设计的出发点是非常重要的，也是非常困难的。像微软这样一个成功的软件公司有时也不能很好地回答它们，不正说明它的困难所在吗？不正说明的确应该下功夫来面对它吗？

那么，应该怎么做呢？

让我们从基础的基础——设计之源开始。

5.3 设计之源

下面根据一些生活中最简单的例子来看看，设计究竟为了什么。

案例分析

图 5.1 中的三个设计来自远古：一个石制标枪头、一双鞋和一个瓦罐。

石制标枪头有锋利的边刃和顶端，用于狩猎。它的设计造型简单流畅，极为实用，是当时最有效的狩猎工具。这是一个成功的设计，是为了生存而产生的一种设计。



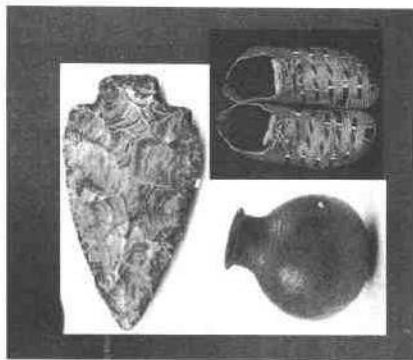


图 5.1 远古的三个设计

这双鞋由兽皮条制成有孔鞋面，不仅通风舒适，而且避免了用整张兽皮制成曲形鞋面的困难，的确是一个精彩的设计。

这个瓦罐设计更是精彩绝伦。它的圆形设计不仅提供了最大可能的容积，而且使制作工艺简单（圆形容器比方形的更易制作），又美观大方实用，真是一个不可多得的设计。只要看看今天同样功能的瓦罐，就会发现当今的设计不过是这一古代设计的延伸。这真是一种历史性的精彩设计。

这三个成功的古代设计告诉我们，设计是为了生存。设计之源在于用户。满足用户的需求，便于用户的使用，同时又使得生产工艺尽可能简单。这就是设计之本。

现在再来看看一个近代设计的例子（如图 5.2 所示）。它再次说明了这一基于用户的设计之本。

案例分析

曲别针是在办公室和家里到处可见的极简单的物件。可方便地用它来夹住几张纸，又可方便地从夹住的纸堆中将一张纸取下来，还可方便地把曲别针从夹住的纸堆上取下来。曲别针完全地实现了我们（用户）对它的需求。而且更为难能可贵的是，它的设计直截了当，形式极为简单，又极为易用、易制，不需要读任何说明书就知道怎样使用它。这就是曲别针沿用至今，经久不衰的原因。

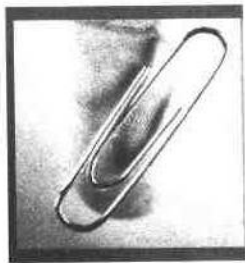


图 5.2 曲别针的设计

历史上曾有很多人试图重新设计曲别针，但都不能改变它的基本设计。若不信，不妨试一下，看能不能想出一个更为简单但又能完成相同功能的设计？会发现这是一个相当困难的挑战。

其实，中国的筷子也是一个相当成功的设计。它的造型极为简单，制作极为方便，却可用于夹取除汤以外的几乎所有的食物，真称得上是一个伟大的历史性设计。

但是，也有例外，请看下面的案例。

案例分析

有一次，我太太从美国的一个超市上买回一双筷子，有大约一尺半长，大大长于一般家用筷子。它用材讲究，制作精美。所以尽管它价格不菲，我太太依然买了一双，说可用它从锅里夹面条。因它很长，可以避免手被锅里冒出来的蒸汽烫伤。

的确，这双筷子是一个有创意的产品。设计者试图把筷子从饭桌上搬到灶台上。可惜的是，它有一个致命的设计缺陷：它的头部突然变细，在使用时两个筷子头不能并拢，很难夹住任何东西（如图 5.3 所示）。

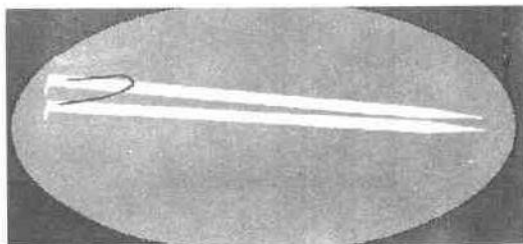


图 5.3 有设计缺陷的筷子

显然，这双筷子的设计者并不真正了解人们是怎样使用筷子的。他不了解人们是用并拢的筷子头把东西夹起来的。这样的设计不遵从用户的需求，与设计之本背道而驰，怎能赢得用户呢？我家的这双筷子已被我太太扔到垃圾箱里去了。

其实，如果我们仔细观察就会发现，违背设计之源的设计在软件生产中和日常生活中时有发生。在细述应怎样设计软件之前，让

我们看看错误的设计是怎样发生的。

5.4 错误设计之源

5.4.1 技术至上

专业软件研究人员，尤其是做基础软件研究的，对最新的软件技术大都有相当的了解。他们相信技术是最酷的，是万能的，是至上的。当他们写软件时，想的是如何用最酷的方式表达他们的技术，而很少思考用户和用户的需求。或者单纯从技术出发，想当然地将用户需求硬安到他们的技术中，而不做任何认真的用户需求分析。正像古人所说，削足适履，“足”为用户需求，“履”为技术。

这真是“本末倒置”！一个软件产品，一定要有人买才能成功。大多数人买软件并不是为了其中的酷技术，而是因为它能满足他们的需求，能使他们的生活、工作更方便一些，更有趣一些。因此，设计软件产品的出发点应该是用户和用户的需求。技术应该被调整以满足用户和用户的需求，而不是相反。简言之，是市场决定了产品的设计，而不单纯是技术。

遗憾的是，这种“本末倒置”的“技术至上”者，在当今世界上比比皆是。我曾在一篇美国的商业杂志上看到一些统计数据。在美国，每年有很多的高新科技公司成立，也有很多的高新科技公司倒闭。倒闭的高新科技公司中，90%以上不是因为技术的落后和失败，而是因为他们不能正确地了解市场。

让我们再来看一个在美国高科技史上的令人痛心的例子。



案例分析

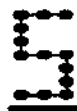
大家知道，DEC 曾经是美国三大计算机公司之一。几年前被康柏收购，从地球上消失，成为美国计算机界一大憾事。DEC 曾以众多的高新技术著称于世。其中，它在最后几年里研发出的 Alpha 计算机芯片更以其卓越的技术在性能上超过了 Intel、Sun 和其他厂家生产的芯片。微软也曾大力协助 DEC，将 Windows NT 移植到 Alpha 系统。然而，Alpha 在市场上彻底地失败了，成为 DEC 最终失败的原因之一。为什么？

究其根本，还是因为 DEC 对其市场和用户不甚了解，盲目追求技术而铸成大错。Alpha 系统复杂，造价过高，绝大多数用户不能接受；而且它又不能与已有的系统兼容，使得在其上的软件开发相当困难，软件开发商亦很难接受。Alpha 把它的两级用户都得罪了，要想成功，谈何容易！

5.4.2 惟我独尊

搞软件的人大多受过很好的教育，都非常有智慧、自信，又热爱技术，热衷于发明，属于高智商群体。正因为如此，我们中很多人，自觉不自觉地把软件设计当成自我表达的一种方式——表达自己的智慧，表达自己的技术，表达自己对技术的虔诚。

但是，不能过度地使用自己的智慧和自信，否则会落得个聪明反被聪明误的结局。



案例分析

有一次，我带一个研究员参加产品部门的一个会议。这个研究员向与会者介绍了他的一项图形技术。产品部门的人说，他的技术很酷，但不适于他们的用户。这个研究员顿时变得急躁不安，质问别人懂不懂他的技术，懂不懂用户；质问别人为什么这么酷的技术对用户没用。总之，他对产品部门的解释不屑一顾，感觉自己在对牛弹琴。

这样的会议对我这个协调者和组织者来说，真是尴尬。这个研究员技术上非常优秀，但他并不懂这些产品，更不懂这些产品的用户，却质问那些最懂这些产品和用户的人。真不知谁是“牛”，谁在“弹琴”。这次会议不仅没能达到技术转移的目的，而且为以后的合作蒙上了阴影。


我想通过这件小事说明两个问题。首先，在设计软件时，不应凭空想象我们的用户，不应想当然，而应对市场和已有的产品做调查、分析。毕竟我们自己对软件的需求，只代表了用户需求的极小的一部分。大多数用户并不像我们这些搞计算机的人那样使用软件；其次，我们应善于学习，向市场学习，向用户学习，向懂行的人学习。真正看懂市场要什么，用户要什么，产品要什么。

请记住，研发人员的自我表达不应成为软件设计的目的！

5.4.3 满足人的一切需要

软件设计的另一个极端是：满足人的一切需要，而忽略技术上的可行性。这种错误通常从一个野心勃勃、不着边际的想法开始，完全从人的需要出发，忽略已有技术的可行性，或过分依赖未来新





技术的突破。

这样的设计通常会导致软件的失败，会浪费大量的人力物力。历史上最著名的一个例子应算日本的第五代计算机了。


案例分析

20 世纪 70 年代，日本政府和学术界发起了“第五代计算机”的革命，投入大量的财力和人力，其目的在于使计算机以人的智能帮助人类。其中包括能看、能听、能说、能与人对话，能理解人及能翻译的智能机器人。这一革命在当时极大地推动了计算机人工智能的发展。可惜的是，人们过于乐观，操之过急。人工智能的发展未能如愿，绝大多数基本技术不能解决。日本最终不得不宣布放弃。“第五代计算机”终成泡影。计算机人工智能这一学科也因此蒙受了不白之冤，被很多人称为无用的学问。

这样的设计也常常使一些技术上并不成熟的产品流入市场，误导市场，而最终又被市场淘汰，例如，若干年前的翻译软件、语音识别 / 理解软件以及现在的人脸识别软件，因技术不成熟，过于超前市场而导致产品失败，公司倒闭，在美国也是屡见不鲜的。

软件设计工作只有基于用户需求，立足于可行的技术才有可能成功。

5.4.4 功能至上



软件的功能指软件为用户完成一件事的能力。例如，语音识别是输入软件的一个功能，语法校正是文字处理软件的一个功能等。如果一个功能真能帮助用户完成一件事，称这个功能是“有用的”。



(useful)”。实现软件的一个功能，通常有许许多多的方法。将一个功能展现给用户，使得用户能使用它，也有许许多多的方法。如果一个功能的实现能使用户很方便地使用，称这个功能是“可用的 (useable)”。

“useful”和“useable”是两个不同的概念。前者强调有用性 (usefulness)，后者强调可用性 (usability)。对软件的可用性研究，通常是对用户心理行为的研究。软件的可用性一般体现在对用户界面的设计上。一个好的用户界面设计，可方便用户的使用，也更易在市场上成功。

有用性和可用性是设计软件产品的两个重要方面。遗憾的是，在传统的计算机教育中，我们更多地强调的是软件的功能（算法、系统……），而很少涉及软件的可用性（软件用户的心理行为 and 用户界面设计）。这是这些专业软件研发人员在设计产品时的“先天不足”，或曰“功能至上”——只讲功能，而忽略了基于用户心理行为的用户界面设计。

这是一个相当普遍的问题。如果不能克服这一不足，就只适合实现一个软件产品，而不是设计一个软件产品。

5.5 基于用户情景的设计

基于用户情景的设计是微软产品部门常使用的软件设计方法之一。这一设计方法的核心思想遵从一个很简单的原则：设计从用户出发，从用户的问题和需求出发，而不是从技术出发。遗憾的是，这样一个简单的原则，常常被设计人员忽略。因为设计常常是由软件工程师做的，而软件工程师常常强于技术而弱于对用户的理解。这一方法可以帮助软件工程师扬长避短，规范设计。

在讨论这一设计方法之前，先看看“用户”在这里指的是什么，“情景”在这里指的是什么，然后再来描述一个典型的设计过程。



5.5.1 用户

在这一节中，讨论怎样解决 3W（Who, Why 和 What）中的“Who”。

大多数软件是为人设计的，但是任何一个软件都不可能完全满足所有人的需求。它一定是为一个特定的人群设计的，如办公软件针对的用户主要是办公室的工作人员。不同版本的视窗操作系统也是面向不同的对象的，如服务器、数据中心服务器、家用客户端及办公室客户端等。这个特定的人群叫做这个软件的用户群。软件设计的第一步就是确定用户群。

一些设计人员常犯的设计错误就是把他的软件想象成是为所有人服务的，或把用户群定义得过广过宽。这是个相当致命的错误。这样的软件即使能编制出来，也是属于那种“什么都能做，但什么都做不好”的和无特色的软件。因此，必须非常明确地定义用户群。

在微软公司，一些产品部门，例如浏览器、办公软件和视窗等部门，使用一种叫“典型用户（Persona）”的方法来定义和描述他们的用户群。Persona 是一种基于市场分析和可用性研究的对软件用户的具体描述。一般地说，设计人员从市场分析和可用性研究的数据中确定他们的用户群，然后从这一用户群中选出或构造出典型的用户个体，并用 Persona 来加以描述。

因此，Persona 可以看成是对典型用户个体的具体描述。它一般包括以下内容（部分或全部）：

- （1）人名（可以是虚拟的，用于软件设计和开发中的沟通）
- （2）年龄（不同年龄段的人可能有不同的需求）
- （3）收入（不同收入段的人可能有不同的需求）
- （4）所代表的用户群在市场上的比例和重要性
- （5）使用这个软件的典型情景（将在下面叙述）
- （6）使用这个软件的环境
- （7）用户的工作情况（在办公室里？常常在外出差？是管理人



员还是决策者……)

(8) 用户的生活情况 (有家? 有小孩? 有不少亲戚? 喜欢外出度假……)

(9) 知识层次和能力 (教育程度, 对计算机的熟悉程度……)

(10) 用户的动机、目的和需要解决的问题

(11) 用户喜欢什么和不喜欢什么

(12) 用户的相片 (用于形象地刻画用户, 便于软件设计和开发中的沟通)

(13) ……

其中, 情景是此人对这个软件的需求的一些典型例子, 将会在下面有详细的讨论。还可以加进其他一些内容, 来帮助理解和描述用户群。当然, 这些内容应该是与软件的市场和设计有关的。

现在来看一个 **Persona** 的例子。假设要重新设计微软在美国的一个网站的用户界面。希望这个新的设计能吸引更多的新用户, 或能使那些不常用我们网站的用户更经常地到我们的网站来, 以增加网站的用户访问量, 并以此增加网站的广告收入。

首先必须从市场分析开始, 看看哪些人有能力却不用, 或很少用互联网; 看看哪些人在使用竞争对手的网站; 看看哪些人是我们的忠实用户。由于篇幅的限制, 在这里不做具体的市场分析, 只列出我们假设的分析结果, 并用 **Persona** 来描述。

根据市场分析, 可以列出如下 4 类主要用户 (4 个用户群, 见表 5.1):

(1) 潜在的互联网新用户。他们很少用计算机, 对互联网有一点畏惧感, 但又觉得互联网好玩, 想试一试, 以丰富业余生活。用 **Persona Alan** 来代表这类用户。

(2) 刚刚开始使用互联网的家庭用户。他们对互联网很有热情, 但担心对小孩有不良影响, 对安全性有顾虑, 因此不常使用。用 **Persona Laura** 来代表这类用户。





(3) 老客户。他们经常使用我们的网站, 主要使用网站的各类

服务。用 Persona Samuel 来代表这类用户。

(4) 年轻一代互联网爱好者。他们主要使用 AOL (我们的竞争对手)。用 Colby 代表这类用户。

表 5.1 的最后一行列出了每个用户群的“设计目的”——对每个用户群的基本设计原则。

表 5.1 四类主要用户群

Vital Stats Chart	Alan	Laura	Samuel	Colby
				
年龄	28	36	61	14
配偶	无	查尔斯	格雷德丝	无
家庭成员	1	3	2 (但有 3 个已成人的孩子和 7 个孙子)	4
城市	波特兰	芝加哥	亚特兰大	丹佛
职业	城市规划部经理	销售助理/妈妈	已退休	学生
使用频率	非用户	偶尔使用	经常使用	青少年
使用范围	面向娱乐	面向家庭	面向职业	内容丰富
互联网经验	初学者	初学者	中下水平	中下水平
访问地点	无	家里	家里/以前工作	家里/学校/图书馆
访问提供商	无	AOL	MSN IA (和 Hotmail 用户)	AOL (和 Hotmail 用户)
来源	OEM (新 PC)	CD	下载或通过 CD, 但使用 Hotmail 账号得到这些内容	OEM (父亲房子里的新 PC)
定义格言	“它看上去很酷, 但我觉得自己像个傻瓜, 所以我远离它。”	“我喜欢使用计算机, 但是我担心泰纳会得到一些他不该获得的东西。”	“我每天都上网。”	“我的朋友都上 AOL, 如果我妈妈允许, 我将整晚在线。”
设计目的	简单	安全	服务	好玩

5.5.2 用户情景

用户情景是用户对软件需求的一些典型例子，常常是软件设计中最重要考虑因素之一。一个情景常常针对一个或几个软件功能。具体地说，一个情景应通过事例说明以下内容：

◀ What——设计要解决哪些用户问题？


◀ Why——为什么需要解决这些问题？

让我们来看一个例子。Persona Colby（见表 5.1 第 4 栏）代表了一个典型的用户群——热衷于社交的年青学生。以下是她的一些典型情景及市场调查和分析。

Colby 的情景

大多数早上，Colby 都准时起床去学校。去学校前，她通常很少有时间去检查她的 AOL 电子邮件账户。在学校里，她仅可以在做科学课作业和进行社会学研究时才使用计算机。图书馆虽然有许多计算机，但要使用的人太多了。


除使用学校的计算机外，在回家的路上，Colby 会去县图书馆的一个部门使用计算机。她先登录到她的 Hotmail 账号查看一些她感兴趣的邮件（有时她喜欢使用这个账号而不是她的 AOL 电子邮件账号。因为她妈妈不知道这个账号，而且这个账号可以在任何地方使用——在学校、图书馆和家里）。然后，她通过雅虎分类为她的英文论文查找一些有关奥斯卡·怀尔德的信息。在下线前，她还查看了一个“Backstreet Boys”网站，看看她喜爱的一个乐队有没有新东西。不管在哪儿登录，Colby 都主要用雅虎来搜索网站，但是她希望有一些东西“更酷”且让她无论在何处都能找到她的密友列表和聊天室。



在家里，Colby 每天要使用几个小时的计算机。大多数晚上，她更愿意在网上与朋友聊天而不是看电视；当然，她也有自己喜欢的电视节目，这些节目她从来不会错过。在与 AOL 连接后，她回复了几个新邮件并检查她的密友列表，看看谁在线。她的一个朋友建议她去看看一个网站，这个网站是他们的一个同班同学制作的。另一个朋友告诉她，她喜爱的一位 WNBA 队员正在 ESPN 网站上聊天，因此，她去了这个网站。几个小时以后，她妈妈要她关掉计算机，上床睡觉。在下线前，她快速浏览了一下 Spin 网站上有关 Kid Rock CD 的最新评论。

Colby 对网上购物确实不感兴趣，因为她宁愿去商场。另一个原因是她没有信用卡，而且她的父母也不允许她使用他们的信用卡。她曾经在线浏览过服装和音乐方面的信息，但是没有找到过她要的东西。与她的大多数朋友不同，Colby 很少玩在线游戏。

这个周末，Colby 将呆在父亲的房子里。她想用他的计算机与朋友们保持联系——但她父亲没有 AOL 账户。她不得不依靠她的 Hotmail——这是件很难受的事，因为她父亲的这台计算机又旧又慢，且调制解调器也很古老了，只有 28.8 kb/s。她试图帮助父亲找到使计算机在线更快的方法，但是在地购买一个新调制解调器前，一点希望都没有。她还必须与兄弟一起共享在线时间并遵守她爸爸的规定，即只能在线一个小时。还有，下午常常掉线（失去连接）也是一件令人讨厌的事。



5.5.3 设计过程

这是一个典型的商业软件的设计过程。微软的一些产品就是这样设计出来的。这一设计过程的结果就是软件说明书。

(1) 收集市场数据，做市场分析。

(2) 确定用户。与用户交流，理解用户，理解用户的工作并与用户建立很好的关系，以便在今后的设计和开发中能经常得到他们的反馈意见。

(3) 建立典型的用户群。通过对用户工作的了解，发现与设计有关的典型的用户群。这些用户群应该描述用户工作中的一个或几个重要环节。

(4) 与用户进一步交流来细化用户群，并写出情景脚本 (Scenarios scripts)。

(5) 建立用户 Persona，可以一个，也可以多个。

(6) 确定软件的主要功能，可以一个，也可以多个。

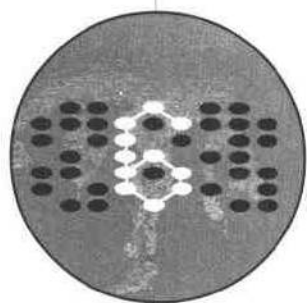
(7) 确定这些功能的主次，并为每个功能确定一个优先权。

(8) 根据典型个体和用户群，设计软件的需求并写出说明书 (specification)。

(9) 用用户群去检查说明书，看看说明书是否满足用户群的要求。

在以上的设计过程中，应该经常与用户保持联系，以便得到他们对设计的反馈意见。

从这个设计过程中，可以看到用户是软件设计中最重要因素。可以毫不夸张地说，软件设计工作中更多的是研究用户和市场，而不是技术。



第6章

项 目 管 理

Program Management

背 景

项目管理是一种广泛应用于各种工程、金融甚至农业生产中的技术管理过程。在IT行业，项目管理常常是决定产品或企业能否成功的最重要指标之一。中国历经了15年的不懈努力，加入世界贸易组织终成现实，这为我们带来了前所未有的机遇和挑战。我国政府所属各部门和企业领导对于项目管理也越来越重视，现在市场上名目繁多的各类项目管理培训就可见一斑。

熊明华在微软担任项目经理这一职位多年。本章是根据他亲身实践，对微软的项目管理进行了详细的介绍，相信对我国软件业项目管理水平的提高会有所促进。

熊明华

熊明华在微软总部的办公室里



微软公司项目经理。参与过 Java VM, Internet Explorer, Windows Me, Windows 2000, .NET My Service 等产品的开发管理工作。目前在 MSN 部门从事项目技术总体设计工作。

1987 年 毕业于长沙中国国防科学技术大学。

1990 年 在北京中国国防科技信息中心获得硕士学位。

1993 年来美国之前，在中国科学院软件所汉京公司担任开发部门经理。

在加入微软之前，熊明华在 IBM 公司位于纽约州的互联网产品部担任高级工程师。

本章内容概览

- ☞ 项目管理简述
- ☞ 什么是项目经理
- ☞ 项目经理的行政结构和工作关系
- ☞ 为什么需要项目经理
- ☞ 项目经理每天的具体工作是什么
- ☞ 做项目经理的背景要求
- ☞ 结论

6.1 项目管理简述

项目管理是一种广泛应用于各种工程、金融甚至农业生产的技术管理过程。在软件和 IT 行业，项目管理经常是决定一个产品或企业能否成功的最重要指标之一。项目管理已经得到越来越多的企业和政府部门的重视。在中国，有关项目管理的首次国际研讨会于 2002 年 4 月召开。学习和借鉴国际上先进的项目管理经验是非常明智和有益的。

软件企业的项目管理规范是许多公司通过近几十年的摸索和实践逐步发展起来的。微软在这方面有许多成功的实例，也有不少失败的教训。微软在 25 年的发展过程中总结并形成了适合于软件产品开发的项目管理经验及一整套的原则、方法和工具。这其中最大的特色就是项目经理（Program Manager，或者叫程序经理）的职位和功能的创立及广泛运用。在微软，项目管理的主要工作是由项目经理负责实施的。为了真正全面地介绍微软的项目管理，必须全面而深入地说明项目经理在微软的具体运作。

本文首先简要地介绍微软创立项目经理的历史，并尝试给项目经理下一个定义。然后将介绍微软软件产品开发队伍中的行政结构和工作关系，并分析微软设立这个专门职位的具体原因。接下来将介绍项目经理每天的工作安排，以及在一个产品开发周期的各个阶段项目经理是如何领导和参与项目管理的。最后特别介绍对项目经理的素质要求，以及微软是如何招聘和考核项目经理的。

6.2 什么是项目经理

小故事

在介绍项目经理之前，给大家讲一个我在 1996 年从 IBM 去微软参加面试的有关故事。那时我在 IBM 的 Internet 事业产品部担任高级程序员，从事网上新闻个性化订阅服务系统（Infosage）的开发工作，同时是该项目的总体设计组的成员。我主要负责用户界面的总体设计，核心模块的编码实现，同时为项目的全球化提供指导。我到微软申请面试的职位是项目经理。当时我以为这个职位应该类似于 IBM 的项目组长（Project Lead）。在 IBM 和其他许多 IT 企业里，项目组长是一个很常见的职位。项目组长一般由主程序员担任，领导开发人员和测试人员，负责技术设计，编写设计说明书，参与实际代码编程，负责产品开发的顺利完成。从微软的项目经理职位说明上看不出有什么与此不同。

当我在微软经过一整天 7 个人、每人 1 小时的面试后，人力资源部的人问我感觉如何，我就马上问：“为什么今天没有人问我任何编程技巧和算法的问题？”她回答说，“项目经

理不用编程。”我当时很吃惊，反问她：“不编程怎么能做好项目经理？”她没有直接回答我的话，只是简单地说：“过几天我会很快再和你联系。”

如今，我到微软担任项目经理已经 5 年多了。这期间我确实没有写过大段程序，然而却参与管理了许多软件产品的开发。我自己对微软项目管理经历了从一知半解到轻车熟路的过程。

微软的项目经理是非常独特的职位，它的组织结构和功能定位在整个软件工业界几乎是独一无二的。微软创建于 1975 年，而项目经理这个职位直到 1984 年才创立。在这之前的近 10 年期间，微软没有项目经理这个职位。其开发人员的组织架构和目前其他大部分软件公司一样。这些软件公司一般由资深的程序员担任技术负责人，直接负责产品的功能分析、模块设计、开发过程、技术决策、重要代码段的编写以及部分测试。项目的日常管理往往以技术负责人为主。一般有一名产品高层经理对整个项目负责，但往往在行政上而非技术上对项目进行管理，并不一定直接参与产品的设计和实现。总之，没有负责产品设计和日常管理的专门人员。另一方面，产品开发组一般也没有专门的测试组人员。项目往往要请产品开发组外的测试人员在最后的开发冲刺阶段进行高强度的测试。

那微软是如何开始设立项目经理这个职位的呢？创立项目经理的原因有其偶然性和必然性。微软创立最初 10 年左右，项目组织结构和其他大部分软件公司差不多。在 20 世纪 80 年代初，随着软件产品项目规模越来越大，产品质量问题开始变得非常严重。产品开发的管理混乱，周期也越拖越长，客户投诉越来越多。微软越来越意识到有必要从根本上加强项目的组织管理。

第 6 章



项目
管
理

微软项目经理的历史

在 1984 年,有一个叫 Sabe Blumenthal 的人,他在 Multiplan 产品组担任开发组长,即技术负责人。他在开发 Multiplan 过程中感觉到有必要有人专职做产品功能的定义、规划和设计。这个人需要做产品开发中的决策,协调开发人员、测试人员和市场部门的日常运作。但这个人不必做实际代码的实现或测试,也不直接做市场方面的调研。他自己试着这种角色,并且成功地完成了这个产品的开发。不久,他把这个想法和经验向比尔·盖茨汇报。比尔非常重视并肯定这一尝试,决定向全公司推广设立项目经理职位。

今天微软项目经理共有 4 000 人之多,将近占产品开发总人数的五分之一。即一般每个开发小组中,有一个项目经理、两个软件工程师和两个软件测试工程师。这 3 个职位可以说是微软开发队伍中的三驾马车。其中项目经理起领头作用,对微软几千个产品开发的成功起了非常关键的作用。

那么,究竟什么是项目经理呢?

项目经理

项目经理在微软是负责并保证高质量的软件产品按时完成和发布的专职管理人员。他的任务包括倾听用户需求;负责产品功能的定义、规划和设计;做各种复杂决策,保证开发队伍顺利开展工作及跟踪程序错误等。总之,项目经理全权负责产品的最终完成。



6.3 项目经理的行政结构与工作关系

在项目队伍中，项目经理是领头人，是协调员，是鼓动者，但是他并不是老板。这是什么意思呢？在微软，我作为项目经理，虽然针对项目开发本身有很大的权利，可以决定进度和功能块，而且软件设计工程师和软件测试工程师要遵守这些决定。但是从行政上讲，他们并不属于我的部下。我并不直接干预和影响他们的工资、奖金和升迁。

在微软，项目产品组的行政结构，是一种垂直性的专业化的管理模式。任何一个产品组，都相当于一个相对独立运作的小公司。这个小公司在行政上是1加3的结构，即一个产品单元经理（Product Unit Manager）加上三个部门经理。他们分别是团队项目经理（Group Program Manager），软件开发经理（Development Manager）和测试经理（Test Manager）。这三个部门经理平级，并都直接向产品单元经理报告。这三驾马车对产品的顺利开发成功起关键的领导支撑作用。另外，每个部门经理可管理几个组长，而每个组长管理3到5个具体工作人员。

例如，从行政上讲，3到5个软件设计工程师归一个软件开发组长管理，几个软件开发组长归一个软件开发经理管理，软件开发经理直接向产品单元经理负责。这些组长和软件开发经理都是软件设计工程师出身。他们都能编程，而且能指导手下的人员编程或代替他们完成任务。同样地，测试人员有测试组长和测试经理，测试经理也直接向产品单元经理负责。测试人员在行政上、技术上、业务上也实行一元化管理。

项目经理也有类似的行政隶属关系。由于项目经理人数相对少，有时直接向产品单元经理汇报。项目经理通过产品单元经理对软件设计工程师和测试工程师的资源 and 任务分配进行调整，但不会直接





下达行政命令。项目经理的核心任务是业务领导，掌握产品全局观念和进度，协调好软件设计工程师和软件测试工程师的工作进度，并使他们相互之间紧密配合，同时与该产品有关的其他部门和人员，如市场、用户支持、客户教育等进行协调。

图 6.1 所示是项目产品组的行政结构图。

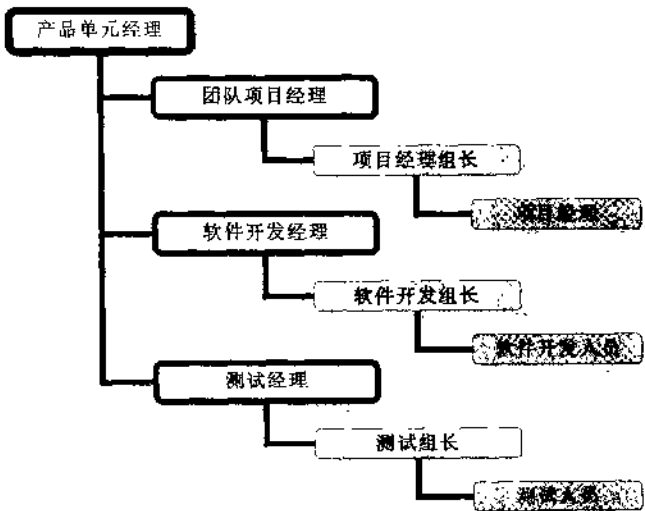


图 6.1 项目产品组行政结构图

采取这种垂直性的专业化的行政管理结构的基本原则有以下 3 点。

1. 产品设计、产品实现和产品测试是三权分立式的相互配合，相互制约，同时又是相互依赖的关系

这是同一个产品开发过程的 3 个方面，它们都在具备非常专业化技能和知识的专门人才的直接控制管理下。三权分立的模式有其科学性和必要性。

首先是专业分工明确，责权分明。假设测试组归属开发人员管理，测试工作将很有可能失去独立性和质量控制的严正立场，更多



地受开发人员的干涉甚至排斥。因为开发人员大都不喜欢被指正程序失误，而更喜欢争论是否测试本身存在问题。类似地，如果开发人员归项目经理直接进行行政管理，开发人员很可能为迎合领导而在开发中尽量接受和满足项目经理在产品设计方面的干预，而不是真正从技术实现上思考和评判设计方面的问题。另一方面，开发人员也会由于项目经理不直接编程，或对自己的具体建议不采纳而产生抵触情绪，因而影响产品开发的正常运作。

总之，这 3 个专业领域只有分工协作才可能建立明确责任制，从而保证产品开发的顺利完成。

2. 人员的行政管理是专家式管理

这种专家式的管理使得微软的任何一级领导都能充分了解手下人员所做的工作并能及时对下属人员提供专业上的指导和帮助，必要时能在下属缺席时替其完成工作任务。下属也会乐意接受能真正了解他的具体工作状况的人做领导，并给予充分的尊重。即员工不会尊敬那些没有能力却当他们所做工作的主管的人，也没有人喜欢只会管人而不懂专业的领导。

许多软件公司在对项目进度评估时，因为老板不懂专业，经常出现开发人员虚报所需工作日的情况，但这种情况在微软很少发生。任何一个开发人员都清楚，你的估算会很直接地受到你的软件开发组长的准确评判；而且软件开发组长的估算在软件开发经理那里也会很容易受到检验。原因很简单，即你的领导能在专业上对你的情况了如指掌。你只能诚实对待你的工作，不大可能欺上瞒下。

微软这种专业合格导向的主管和明智的管理模式是微软成功的核心思维和企业文化之一。

3. 专业人才的发展遵循不同的业务具有同等的提拔机会的原则

这为专业人才的事业提供了广阔的发展空间，避免了人才使用不当的问题。在微软，行政级别决定员工的工资、股权等福利待遇，





而职务头衔与行政级别并不是直接挂钩的。

例如一个程序员，只要能力强、贡献大，其行政级别及他的工资、福利待遇等可能与总经理甚至副总裁一样，他不需要成为经理或管人的领导才能获得较高级别待遇。同样，无论项目经理还是开发人员或测试人员，其行政级别都可以达到较高水平。如资深的测试人员可以在行政级别上得到提拔，开发人员也可安心研发软件。他们的能力和贡献能通过提级不断得到认可和回报。

以上主要分析了项目经理的行政结构关系。下面将分析项目经理与产品组的其他人员的工作关系。其实这可以归纳为产品开发人员之间的横向关系。

在微软，一个稍大点的产品部门，比如我曾经长期工作过的网络浏览器产品部门，分成几个功能块组（Feature Team）。每个功能块组一般负责一个具体的功能模块，并由 10 到 50 人组成。根据功能模块的大小，功能块组可能被分成更多的子功能块小组。最后每个小组一般不超过 10 人，其中至少会有一个项目经理，几个开发人员和几个测试人员。这组人员依据功能模块组成一个工作小组，但是所有成员在行政上并没有直接上下级关系。他们共同负责具体设计、开发、测试并最终完成这个功能模块。一个项目经理可能拥有一个或多个功能模块。一个开发人员或测试人员也可能被分配到多个功能模块组中。

这是一种交叉的工作关系。对于任何一个功能块而言，一定有一个项目经理领头，从设计和管理上带领一组开发人员和一组测试人员共同负责。三种角色缺一不可。这是一个以项目经理为中心、以功能模块为基础的工作关系。

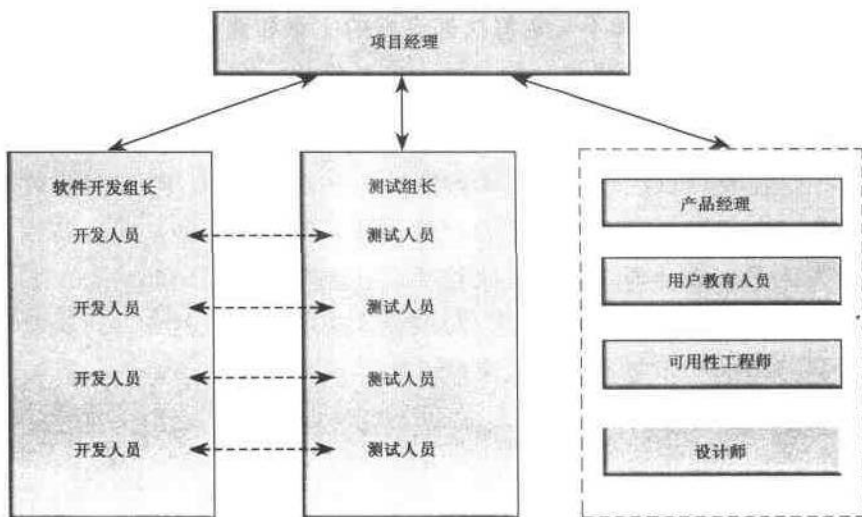
图 6.2 所示能比较清楚地表明这种关系。

从图 6.2 所示中，能清楚地看到项目经理除了直接与开发组和测试组密切联系以外，还直接与产品经理、用户教育、可用性工程师和设计师等进行联系。这些人员也是通过项目经理对产品功能的具体设计、技术实现进行指导并提供帮助的。



如果项目经理不从行政上直接领导这个工作小组，那他如何能有效地进行项目管理呢？其他人员为什么会听从项目经理管理呢？

在微软，项目经理的威信只能是通过努力工作，从实践中逐步赢得同事的信任而建立起来的。不是所有的项目经理都能成功地干下去。不少人因为不能赢得开发人员和测试人员的尊重和配合，而不得不离开某个产品部门或转向其他不同的职位。项目经理的权威来自于或者说形成于他自己工作中表现出的较高的领导力、洞察力和判断力，以及高素质的技术专长和出色的沟通协调能力等。这种权威的形成是自发的和自觉式的，也只有这样的权威才是更有效的和长久的。



在项目的开发过程中，项目经理的领导能力是非常重要的。项目经理必须为项目组高效率的运作提供强大的推动力。比如，负责召集并主持每周的项目进度会议，制定会议议程，确定工作事项并进行实时跟踪，还要提交项目进度报告等。项目经理经常需要在错综复杂的情况下迅速洞察问题症结所在，并做出好的判断和提供解



决对策。项目经理还应具备高素质的技术专长，能真正帮助开发人员解决设计上的难题，能帮助测试人员分析和正确判断程序错误，能充分发挥和运用可用性工程师、用户教育人员和设计师的专长。项目经理要让他们感到你能理解并尊重他们，从而赢得大家的信任。当存在技术设计或项目管理的矛盾冲突时，项目经理应根据充分的材料和数据，合情合理地最后的选择方案做出决定，而不是凭空臆测或刚愎自用地进行决策。一旦你在项目组中有着广泛的尊重和信任，或者说好的口碑，开发人员会经常主动与你讨论设计问题，测试人员也会尽最大努力配合你的工作进度安排。

另外有两点需要特别提出来给予说明。

(1) 项目经理不一定每次都有好的主意和建议，而要善于倾听大家的好主意和好建议，能帮助大家总结出一个大家都信服的好结论。

这需要项目经理具备好的判断能力和组织能力。项目经理在产品阶段并不是关起门来自己做设计，而是与其他队员，特别是开发人员、设计师、可用性工程师，甚至产品经理一起讨论，归纳出大家的共识，再形成文字，写成设计说明书。当然这些讨论是以项目经理的初步方案为线索来展开的，项目经理要引导大家并发挥各专业人士的特长。

案例分析

前不久我被指派去处理 .NET My Service 中的一个非常棘手的技术设计难题，即如何支持数据排序与存取的全球化和个性化。My Service 的后端是上百个大型数据库，存有上亿个来自世界不同国家或地区，不同语言的用户个人数据。



每当一个用户存取数据时，如何根据这个用户的语言或特别要求，自动地提供符合该用户排序习惯的结果？例如，中国用户查询自己的通讯录，可能希望结果按拼音排序，但是香港用户希望按笔画排序。作为后端的中心数据库不可能按单个用户的要求存储这些排序信息。我的任务就是提出一套设计解决方案来支持个性化的服务。

接到这个任务后，我迅速找到微软内部在数据库、操作系统方面曾经负责排序处理的有关开发人员，寻求他们提供确切的技术实现情况，同时向.NET My Service的核心数据流的项目经理和开发人员了解目前的实际技术问题，另外还要联系微软在日本和其他地区的负责本地化的技术人员，以便进一步确立用户需求。最后把这三方面的数据和建议汇总、分析并提出解决方案，形成详细设计说明书。再把说明书初稿交给所有有关人员进行审阅，在此基础上完成最后设计。在完成设计后，还必须与具体负责此模块开发的人员进行详细讨论，以便写出技术实现说明书。还要和测试组人员沟通，以便开始进行测试计划的制定等。

总之，为完成这项设计，必须和许多不同部门的开发人员或项目经理进行交流，集思广益地把大家好的意见和建议总结出来。

（2）项目经理要学会使用开发人员和测试人员的共同语言。

我们知道开发人员作为一个群体，他们的语言是有一定特性的。如果不能观察到这种特点，就很难与他们充分沟通，他们也不会喜欢与你交流。

案例分析

开发人员在研究和讨论设计实现或分析程序错误时，一般倾向于谈论到应用程序接口（API）的细节。有时，项目经理不能真正深入到应用程序接口的参数的设计、定义或使用，使得开发人员感到没法真正讨论下去。开发人员认为无法从项目经理那里得到有效的帮助，从而避开与项目经理开会或接触。

其实开发人员并不期待直接从项目经理那里得到所有的技术设计与实现的答案，他们更多地想与项目经理讨论问题，使他们自己的很多想法得到项目经理的反馈，特别是想知道这些想法是否与功能设计或需求分析一致或冲突。

越是技术强的项目经理越容易与开发人员展开讨论和沟通，也越受到欢迎和尊重。在微软，也有些技术背景不够强的项目经理，他们感到不受开发人员尊重，害怕直接与开发人员接触，因而不但不更多地从事事务性的项目管理工作。

另一方面，测试人员必须读懂项目经理写的功能设计说明书，以便制定测试计划和设计测试案例。然而，功能设计说明书往往不容易读懂，或者细节不一定足够详尽到让测试人员直接着手工作。这就要求项目经理向测试人员进一步解释，并提出可能的测试情形分析。譬如在哪些先决条件下，可能出现哪几种结果，其中哪个结果是设计所预期的。项目经理一般会邀请测试人员参加设计说明书的评审会议；测试人员也会邀请项目经理参加测试计划书的评审会议。在这些会议中双方要积极理解对方的说明，并提出建议，使设计和测试能相互一致并且没有遗缺。

当然，项目经理在与设计师、用户教育人员、可用性工程师以

及支持工程师一起工作时，也必须对他们的语言习惯有相当程度的了解，以便能真正交流起来。总之，交流技巧是项目经理必备的条件之一。关于项目经理的其他必备条件或素质，在稍后还会有更多的叙述。

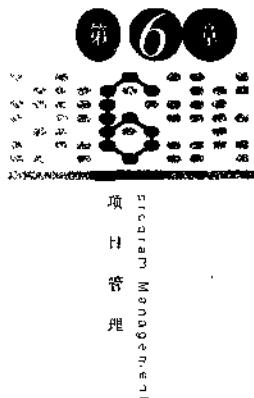
6.4 为什么需要项目经理


微软的项目经理的独特性主要体现在什么地方呢？为什么一定需要设立这个角色呢？可以首先从一般软件公司项目组织结构的分析入手，观察这种普遍存在的结构的主要问题是什么。

在大部分软件公司里，一般是采用以技术负责人为主体的开发组织模式，即一个或几个主要技术负责人带领开发人员和测试人员组成的模块小组，具体负责该模块的设计、实现和测试。技术负责人参与项目的计划制定，需求分析和管理，项目的日常跟踪，同时也会承担高层设计、编码以及小组人员的协调及行政管理。技术负责人往往直接向产品总监或总经理报告。产品单元经理主要负责项目的定项、预算、人事等，但往往并不参与项目的日常管理或技术指导，他们更多地是从行政、人事和财务上管理这个项目。这种结构存在的主要问题有以下几点。

(1) 一方面，项目负责人的职责以管人而不以管项目设计和实现为主。这些负责人一般不容易得到项目组的技术人员的真心尊重。大家的积极性也不容易调动起来。另一方面，技术骨干往往只能通过成为管人的负责人才能被获得提拔，但他们往往更喜欢做技术工作而不是管人或管事性的工作。

(2) 很多资深开发人员或开发组长除了编写程序代码，还需要负责设计说明书的编写，以及程序错误的跟踪与解决方案的协调等。他们不能真正集中精力解决技术实现上的大量分析和设计任务。开发人员容易也愿意专心编写代码，但不一定对程序错误反应及时，也较容易失去对错误控制的跟踪。开发人员一般不愿意花大量时间





坐在会议室里，他们多半不喜欢也不擅长做各种产品的演示报告。


(3) 没有人真正感到“拥有”这个项目或产品。所谓“拥有”这个产品和项目，指真正全权对这个产品的成功完成负最终责任。在很多项目组里，好像每个人都有一些责任和任务，但又觉得主要是别人的事情，很多情况下仅依赖于个人兴趣或行政上领导的压力。软件开发组长承担大量的设计和实现任务，但他往往对测试不能直接控制。而项目的最高负责人往往在行政上掌控项目，并不能从日常运作上管理项目。总之，没有一个人完全对产品或某些功能的成败负全责。

(4) 当开发人员超过一定规模，譬如 5 人以上时，专职的项目经理显得更加必要。项目规模越大，这种需求会更加明显。否则功能块的协调和合作，进度的制定与落实等会变得容易失控。这些工作没有专职的经理级人员负责，就会分摊到技术负责人或资深程序员身上，或者根本没有人负责，从而导致项目进度拖延，预算超支，甚至项目取消。

微软的 1 加 3 式的项目组织管理结构突出了专业化的分工与协作，同时避免了人才使用不当的问题。项目经理是微软几十年来成十上万个软件产品开发过程中总结和证明了的行之有效的管理组成模式。它的设立和存在主要满足了以下几个具体需求。

(1) 开发人员能够集中精力做开发，而不被管理琐事所困扰。

项目经理承担的管理琐事是非常多的。组织召开会议就是具体的日常任务之一。在微软，虽然控制会议的规模和次数方面比较有章法，但毕竟是一件烦琐的事情。一般项目经理每天会组织、参与各种与项目有关的会议，会前要准备议程，会中主持进度，会后要写总结报告并跟踪。开发人员如果主持或参加此类会议，将严重影响其编程进度和效率。其他的管理琐事还包括审查所有程序错误，处理项目运作过程等。总之，开发人员需要项目经理来帮助他们集中精力写高质量的代码。





（2）开发队伍需要有视野良好的领导。

一个开发组中，如果大家都整天埋头编写自己负责的模块代码，最终可能做出一个高质量而不一定符合公司整体战略，或市场用户真正需求的产品。程序测试人员倾向于发现更多的程序错误，对质量控制的精确和尺度不太能有全局和商业视野。项目经理需花专门时间对产品的战略定位，用户的需求获取和日常管理，以及与其他公司产品的相互配合进行认真研究。项目经理对功能的优先级处理，程序错误的判断决策都可以进行专业化管理。

（3）项目组内部不同角色人员间需要好的沟通协调。

项目组内有程序开发、测试、用户教育及可用性研究等方面的人员。这些人员对项目的过程和具体设计经常存在分歧与冲突。比如，测试人员倾向于发现更多的程序错误，对一些程序错误的理解和认识往往与程序开发人员并不一致。测试人员也可能对产品功能的测试重点不能准确把握，对质量控制的精确和尺度不太能有全局和商业视野。很明显，这些不同职能人员之间的协调一致和相互配合（甚至妥协），都需要项目经理的积极参与和领导沟通。

（4）工程开发和商业运作之间的差别需要有人从中进行连接和平衡。

一般程序开发人员倾向于做纯技术的事情，喜欢用最新的技术和复杂的设计来表现他们的能力。但是他们往往忽视现实的用户需求和商业利益，不太关心非技术性因素对项目进度和产品的影响。另一方面，产品市场人员往往更倾向于在最快时间内拿出用户最需要或最有竞争力的功能产品，他们对技术实现的困难度和复杂度没有直接认识，也不能理解开发人员的思维和运作方式。项目经理的任务之一就是解决这两者之间的矛盾，使他们的需求得到平衡。

(5) 开发队伍与外界的联系沟通需要有专人进行管理和协调。

项目经理除了要协调项目组内部的工作外，还对外代表项目组进行产品说明展示。外部对象可能是公司内其他产品组，或者是公司以外的合作对象或竞争对手。开发人员很少有做演示的才能，他们也不喜欢做这些。而如何将本产品组的最新状况呈现给外界是极其重要的。精心准备对外交流活动，还是仅建立在随时沟通的机制上，产品组的生存与否有时都会与此有密切关系。所以，产品组的外部环境，以及可能带来的风险管理都需要项目经理的参与和投入。

项目经理最重要的职能就是协调好各方面关系。他要保证项目各部门人员每天都在做正确的事情；所有的决策、设计文档、资源分配、方向选择、目标远景、计划和队伍的精神状态等，都处于最佳运行状态。所以，作为项目经理的领导艺术，在于搞好各方面的关系。我刚到微软不久，曾经看见一位项目经理在办公室桌上放了一本《厚黑学》。我当时很吃惊。现在回头来看才有所领悟，技术和关系两者都必须足够强才能做好一个称职的项目经理。

自 1996 年加入微软以来，我对项目经理的角色的认识从一无所知，到一知半解，再到轻车熟路，是一个成长过程。今天，我已经切身感觉到设立这个角色的科学性和必要性。一些美国软件公司已经开始学习设立这一职位。微软也通过 Microsoft Solution Framework（微软解决方案架构）向软件业推荐这一角色。我希望越来越多的中国软件企业能吸取这一经验，少走弯路，实现跳跃式的发展。

6.5 项目经理每天的具体工作是什么

项目经理每天的工作非常紧张，最明显的形式有两多：会议多和电子邮件多。我有时每天要主持或参加五六个会议，收到四五百个电子邮件，其中需要处理的有上百个。其实还有一多，即需要审



阅和跟踪的程序错误多。项目经理需要对这些程序错误进行判断,设定优先级,严重大,有时要把它转交给其他测试人员进行进一步的测试或修正,有时要把它交给设计工程师进行调试和分析,有时要召开会议进行讨论,并决定是否必须在本次版本发行中纠正或推迟到下个版本去做。

一般而言,项目经理具体执行以下任务:

- (1) 制定产品的远景规划,写出项目规格说明书。
- (2) 制定工作详细任务表,跟踪这些任务的执行情况,保证其符合规格说明书的原始设计。
- (3) 组织会议,评审程序错误。
- (4) 指导项目开发的过程设计和实现。
- (5) 对各种具体实现方案进行取舍并做出决定。
- (6) 协调各组人员之间的交互配合,包括开发工程师、测试工程师、产品经理、用户教育和本地化人员。

如果从一个产品的开发周期来看,项目经理的作用会更加清楚。一个产品的开发周期包括计划阶段、设计阶段、实现阶段、稳定阶段和发行阶段。

在产品的计划阶段,作为第一步,当确定一个商业机会后,产品经理会同产品单元经理、高级项目经理等制定出一个粗略的商业计划书,然后交给项目经理们去准备项目计划草案,包括远景描述、主要功能、建议的时间表和里程碑及所需资源估算等。

第二步,项目经理主持由开发骨干、测试骨干参加的脑力激励式的讨论会,对项目计划草案进行全面细致的分析讨论。这种会议要进行多次,直到主要功能块被大家认同和确立下来。对悬而未定的问题分配给某些具体人员去继续研究。

第三步,项目经理开始着手写一页纸规模的设计说明书,包括对功能优先级的制定并说明理由,对资源进行估算,对时间表进行估算,还要进行风险评估和与其他功能块关系的说明等。

确切地说,一页设计说明书其实是功能块的计划评估说明书。



它并不针对功能设计本身提出任何项目说明，更多地是评估实现这个功能块的成本、目标和条件。比如，在我曾经工作过的 MSN Search 产品组，一页功能说明书有一个专门的模板（template），大致包含以下内容。

功能块说明书模板

(1) 名称 (Name) : <功能块名称>

(2) 队员 (Team Contacts) : <PM, Dev, Test, Editorial, Site Manager>

(3) 回报 (Return)

- a. 目标 (Goals)
- b. 目标回报 (Return on Goal)

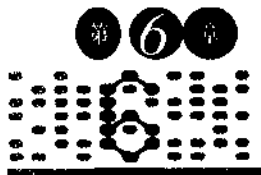
(4) 投入 (Investment)

- a. 工作范围 (work scope)
- b. 主要可行性因素 (Top Viability Issues) : <列出最主要数据、技术、运行，以及外部依赖因素>
- c. 持续进行的工作 (Ongoing work) : <网站管理/数据流程/编辑费用>

(5) 设计 (Design)

- a. 需求 (Requirements)
- b. 用户场景 (User Scenario)
- c. 特性描述 (Feature Description) : <附流程图说明该功能如何与其他功能相互作用>

最后，产品单元经理、各部门经理要参加由项目经理主持的设计说明书的评审会，对有关估计进行调整。最终将所有功能块的一页设计说明书汇总，进而对整个项目的进度、功能块取舍做出总结



和决定。在这之后，项目进入设计阶段，项目经理要着手写详细的设计说明书。

在进入实现阶段和稳定阶段后，项目经理开始担任项目的日常跟踪管理的角色。其典型的工作任务有以下几项。

(1) 主持召开每周固定时间的项目会议。大家相互简报各自领域的进度、状况，特别是存在的问题。开发人员、测试人员都参加这个会议。

(2) 项目经理编写每周的项目进度报告，并发送给项目组全体人员，包括项目组以外的有关人员。

(3) 项目经理主持每天的程序错误评估会议，与软件开发组长和测试组长进行讨论，跟踪程序错误的进展情况。

在进入发行阶段后，项目经理的任务更为紧张。这时每周一次的会议和进度报告改为每天一次，每天的错误评估会改为上、下午各一次。很多问题需要几小时内做出决策。

另外，值得一提的是由项目经理主导的项目作战会议。这个作战会议由项目的核心人员组成的作战小组参加，定期开会讨论项目中所有的热点问题，进行项目的风险评测，决定项目进度的调整或功能块的取舍，同时对市场和客户等有关重大问题进行讨论并做出决议。

在产品完成之后，项目经理必须做的一件事是参与或主持项目的总结报告会。在微软，这个报告会是必须的，而且进行得非常严肃和规范。微软每个项目组在完成一个版本的产品发行后，会及时举行面向全体项目组成员的总结会。项目组的各个小组包括开发组、测试组、项目经理组、用户教育组等等都分头准备，总结出在整个开发过程中做得好的和不好的方面。整个大项目组和各小组举行一天到几天的专访会议。会议的重点在于如何在下一个版本开发中进行改进，而不是找出谁的问题进行批评。每个人要诚实地对自己和别人提出好的和不好的意见。在会议结束时，一定会列出行动列表，并规定好谁去跟踪解决。



6.6 做项目经理的背景要求

下面用微软真实的有关项目经理的招聘广告来观察微软项目经理的主要任务和要求。

微软招聘项目经理的广告

(1) 工作任务包括需求收集(客户和内部产品组),功能定义,开发详细的功能设计说明书,协调开发、测试,用户教育等各方面涉及此功能的人员的工作。

(2) 这个角色要求具备商业发展与战略的扎实技能,能将商业需求转化成技术规格说明书,为整个产品组设定方向,并有高超的交流技能。

(3) 该职位要求候选人具备优秀的口才及交流能力,同时具有高超的写作能力。理想的候选人还必须具有跨不同产品组和外部业务伙伴的工作经验,对政策整合有很好的认识,有在国际环境与政府部门打交道的经验,还具有微软在线产品和服务的经验。

(4) 该职位必须具备很强的沟通技能和有效的公开演讲能力。需要候选人有活力、有智慧、有管理大型项目的能力,有协作能力、伙伴开发能力和关系建构能力。很强的写作能力也是必需的。

从以上这些真实的项目经理招聘广告中,可以感受到作为项目经理的一些基本素质和工作性质。

在微软,项目经理内部也是有不同分工和职责的。最常见的分



为两种，一是以技术设计为主要任务的功能性项目经理，二是以过程管理为主要任务的发行过程项目经理。

我们知道作为最常被人误解的职位“项目经理”并不管人，而是管理产品中的功能模块设计，以及产品的开发过程和进度。那么什么样的人适合做项目经理呢？要求具有怎样的背景和素质的人来担任项目经理呢？

一般而言，在聘用项目经理时，非常注重以下几个方面。

(1) 对生产软件产品有激情，具有领导才能，并对产品有很强的拥有意识。

(2) 对产品设计有强烈的兴趣，同时对技术问题能透彻理解。

(3) 有敏锐的时间和日程观念，能对复杂的问题或任务紧密跟踪，并能区分优先次序和轻重缓急。

(4) 总能充分利用各种渠道和方法来分析和解决问题。对可利用的资源能充分把握。

(5) 有能力迅速而胸有成竹地做出决定，并能做出适当合理的取舍。

我记得第一次微软人力资源部的人打电话“面试”我时，问的大部分问题都与以上几个方面密切相关。例如，“在过去做过的所有产品项目中，哪一个是你觉得最自豪的？为什么你认为这样？”，“你解决过的最难的技术设计问题是什么？你为什么采用那种解决方案？”，“你有什么项目是按计划的时间完成的？没能按时完成的原因主要是哪些方面？”等。

微软在真正面试候选人时，是非常严格和规范的。每位候选人都会被一组面试人员一对一地单独面试。每人时间为一小时。面试后每人必须尽快把面试的评语通过电子邮件发送给所有面试小组成员，以供后面的人员参考。评语的编写也有一定的格式，第一句话必须说出你认为此人被聘用与否，不允许回避或模糊回答。评语中要详细说明面试中你发现的候选人的强点和弱点。

一般而言，如果有两个面试人员的评语是反对聘用，后面的面



试就会停止。如果绝大部分面试人员同意聘用，有可能会增加几个面试人员以便考核候选人是否有潜力担任更重要的职位。

在面试项目经理时，会经常考核候选人是否有在技术和商业之间平衡取舍的技能。一方面，项目经理和开发人员密切合作，而开发人员一般倾向于用最新或最酷的技术来实现产品的功能；另一方面，市场和产品人员则倾向于最快地满足客户和市场的最大需求，甚至不计成本或实现的难度。项目经理需要同时与两者打交道，使得两者的需求和倾向得到平衡。

记得在我被面试的过程中，就有一个有关修改建筑计划方案的问题。这个问题的大意是，一座正在建设施工中的别墅已经进行到一半，突然房主觉得应该有一个小图书室，但房主并不想增加任何成本。请问你如何解决这个问题。这种问题并没有一个标准答案。考核的目的在于了解你的分析问题能力。通过观察你的思路和方法看你是否具有做项目经理的素质。

我当时首先提出 3 个可选方案，包括把客厅的部分空间用于图书室，或者楼上的一个备用客房改为图书室，或者把车库加大用于图书室，同时我还大致说了这几种方案的优缺点。面试人提出几项条件后，我又把这几种方案进行优选顺序排序。但面试人又进一步编出技术上、工程上、成本上的各种理由，一一否认每个方案的可能性，问我怎么办？最后我只好说，如果确定不可行，我就必须跟房主谈谈，考虑放弃这个想法，或者增加资金投入。



在面试项目经理时，也会非常重视候选人的技术设计才能。根据产品组有关的技术背景，会针对性地考核候选人的功能设计能力。

以下是一个有关微软内部面试项目经理材料中的评语样本。可以从中看出对项目经理的大部分基本要求。

微软内部面试项目经理人的评语样本

建议：雇用。总体上，我觉得该候选人具备担任初级项目经理的基本技能。他能从总体目标水平上结构性地思考事情，同时能够合理地把握细节。他表现出了很强的理解能力。他似乎能够做出取舍（虽然我认为他掌握的分寸有待提高）。他有好的技术理解力。同时，从目前已有的反馈来看，他也有很强的设计能力。他还需要证明表现出更多的领导才能——他这方面经验不足，从而不是自然而然地成为一个强有力的推动者。但是，他是一个有智慧的人，有技术理解力，思路清晰，设计和问题解决得很好。我认为他能成长为合格的角色，特别在一个强有力的经理带领下更有可能。

我们开始面试时，谈论了表格产品下一版本的目标。他成功地和适当地简述了这些目标，包括纠正用于去除性能杂块的用户界面，充分利用了 Windows 95 的核心功能，如 MAPI 和 OCX。

我接下来让他细化这些目标，他做得很好。特别是他说出了一个他已开始的工作的特性，而且我感觉他对设计中要解决的问题能把握准确，并提出了一个合理的解决方案。

为了考核理解力，我让他把 MFC 描述给他的祖母听。他立刻问到假设他祖母知道多少计算机的有关知识，同时意识到何时他会遇到她很难理解的概念。他对这个问题的回答比

平均水平高。我接着让他向我描述 MFC，奇怪的是，我认为他在向他祖母解释时，做得更好。另外，我提了一些问题，看他真正知道多少关于 MFC 的情况。当然，他知道得够多了，所以我能看出他具有技术专长。

在结束面试前，候选人和我讨论了一个与 VB 有关的程序错误的评估情形。他能立刻反应并做出我们不需要纠正这个程序错误的建议。这么快的反应令我担心。但是我和他事前进行了角色分工。我作为测试组长，他作为项目经理。应该说他合理和有效率地做出了决定。

他有很好的理由做出各种选择，问了很多好的问题。他花费了一段时间明白了我要他来承担决策的任务，而不是让高层经理或其他人承担。

该候选人性格有一些内向，他需要加强锻炼以便成长为一个强有力的领导者，但我感觉我们应该冒这个险。即他的聪明可以帮助他克服这个弱点，特别在合适的管理者带领下更是如此。

我建议先让他参加一个需要更多设计任务的项目中，而不是涉及太多项目的管理。因为我不知道他是否会自然地沉潜到项目管理中，除非他觉得对设计有一定的责任承担。

通过以上评语，能更具体地看到项目经理的基本素质要求，这包括领导能力、设计能力和理解能力等。

6.7 结论

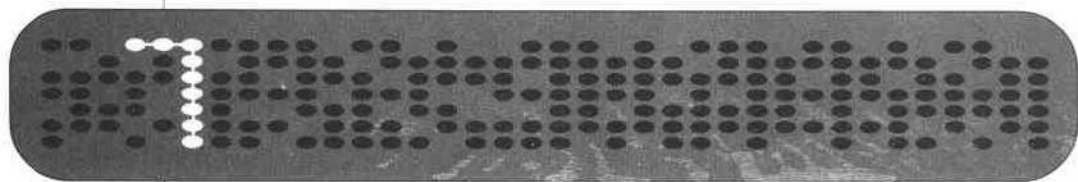
项目经理是微软实现有效项目管理的重要保证。项目经理对产品的成功与否有决定性的影响。项目经理作为一种职业，已经逐步被很多软件公司意识到，并有意识地学习和运用。当然这要结合各软件公司的文化和历史情况进行必要的调整和培养。要真正落实专业化的项目管理，企业本身必须建立与之相适应的企业组织结构、企业文化主流，以及企业项目实现过程。祝愿中国软件企业能尽早达到国际先进水平。中国软件企业应当会大有作为！

第 6 章



项
目
管
理
Project Management





第 7 章

写好代码的十个秘诀

10 Things You Can Do To Write Better Code

背 景

在现代软件的开发中，如何写出具有正确逻辑而且执行速度快的代码是众多的软件开发人员所追求的目标。林斌在微软总部担任了多年软件开发高级工程师，参加了微软多种产品的开发工作。本章，他根据自己多年的亲身体验，为读者总结出一流代码应该具备的特性，以及写一流代码的十个秘诀。文中展示了丰富的具体代码实例，并进行了详尽透彻的分析，最后提供了正确的解决之道。如果你是一名软件开发人员，相信本章对你今后的编程工作会有很大的帮助。

林 斌



微软亚洲研究院新技术开发部经理。

1990年 毕业于中国广东中山大学，获得学士学位，后在美国Drexel大学获得计算机专业硕士学位。

1992 - 1993年 在宾夕法尼亚大学担任工程师。

1993 - 1995年 在自动化数据处理公司 (Automatic Data Processing, Inc.) 任网络工程师。

1995年加入微软公司，先后在不同部门担任软件开发高级工程师 (SDE)，在互联网商业服务组担任软件开发组组长 (Dev Lead)。

1998年以来，林斌在交换协议组担任软件开发组组长。

2000年6月加盟微软亚洲研究院。

本章内容概览

- 简介
- 编写代码的十大秘诀
- 结束语

7.1 简介

自从我在微软中国研究院（MSRCN，现为微软亚洲研究院）工作以来，很多人向我询问在微软进行软件开发的问题，诸如：微软是如何这么快地开发出 Windows 2000, SQL Server, Office, Exchange, MSN 等大型软件系统的？当开发这些软件系统时，每一个开发者如何才能独立工作的同时保持与其他开发者的同步？如何跟踪开发的进程和工程的质量？怎样确保每一个开发者都能写出好的代码？

在研究院工作的这段日子里，我和很多研究员、副研究员、软件开发人员，还有从各个学校来的学生、访问学者一起工作，从中学到了很多东西，同时也发现了不少的问题。这些问题都是有关代码的问题，如代码写得不是很好、不是很清楚。根据这些问题，以及本人以前进行软件开发的一些经验，我总结出编写代码的十个好方法，希望对大家将来写代码能有所帮助。

要写好代码，成为写代码的一流高手，首先要认识下一流代码的特性。

一般来说，一流代码都具备以下特性：

（1）稳定可靠（Robustness）

代码写出来以后，一定要能够运行得非常好，非常稳定可靠。在



现今的 IT 行业，电子商务是一大热门。电子商务运营的关键是 24×7 ，即要保证系统在一天 24 小时，一星期 7 天中都可以无间断地正常运行。我们开发 Exchange Server 的过程就是一个很好的例子。在开发过程中，专门有一个测试组来测试软件的稳定性。这个测试组做很多稳定性的测试工作，包括写一些仿真模拟测试工具。这些工具模仿成千上万的用户通过 Exchange Server 收发邮件，看 Exchange Server 不能在高强度压力下很好地运行，正常地收发邮件。同时他们会定义一系列的性能标准 (Performance Guidelines)，如一秒要收 40~50 个 E-mail。如果 Exchange Server 达不到要求；或者第一天运行正常，第二天就运行不起来了；或者第一天能够收 40 个 E-mail，第二天、第三天就只能收 10 个、5 个，则表明这个 Server 的稳定性不足。到了开发的后期，我们大部分的时间都花在这个上面，尽量提高 Exchange Server 的稳定性。

(2) 可维护且简洁 (Maintainable and Simple Code)

在写代码时，首先要考虑的是：写出来的代码不但要自己可以读懂，而且我们的同事、测试工程师都可能要修改这些代码，对其进行增减。如果代码很复杂，不容易读懂，如程序中的递归一大堆、程序不知何时或从何地跳出，则会使程序的可维护性和简洁性降低。

(3) 高效 (Fast)

在软件行业中效率是非常重要的，比如搜索引擎。有些软件的搜索效率就不高，搜索过程特别缓慢，让人难以接受。当然这里面有一个带宽的问题，但是程序效率不高也是一个重要的原因。为什么雅虎 (Yahoo) 的搜索引擎运行得很快，而且它的准确性很好呢？一方面是因为雅虎专门将搜索结果进行了分类收集，另一方面就是因为其搜索引擎的效率非常高，该程序能够很快地处理用户的请求，并在很短的响应时间之内及时地将结果反馈给用户。





(4) 简短 (Small)

这方面大家的感受可能不是很深，但是我的感受是很深的。我曾和研究院的一个小组开发了一个很小的动态链接库 (DLL)。这个 DLL 实现的功能并不多，它只是从一些图像中提取一些特性（如直方图，以及各种模式、纹理），以便将来在搜索图像的算法中使用这些特性。这个 DLL 中有一个非常大的静态哈希 (Hash) 表，这个表本身就占用了 1.5 MB，使得最后产生的 DLL 大小为 1 MB。在实际使用中，这个 DLL 运行得很好，效率也较高。但是产品部还是不满意，认为这个 DLL 中静态 Hash 表不符合要求。因为这个产品的目标是让用户能够自动从网上升级，这个 DLL 放在服务器上，用户每次登录时，系统如果发现服务器上有更新版本的 DLL，就会把它自动下载到本地，覆盖本地原来的 DLL，因此这个 DLL 就相当于一个自动升级的服务 (Service)。我们都知道，当客户使用 56 kb/s 的 Modem 时，下载 1 MB 的 DLL 需要将近 10 分钟。这么长的升级时间显然是客户不能忍受的。后来，我们对这个 DLL 进行了改进，将静态 Hash 表变成动态生成的 Hash 表，结果使 DLL 的大小由 1 MB 减小为几十 KB。客户在升级 DLL 的过程中等待的时间就大大缩短了。

(5) 共享性 (Reusable)

如果做大型产品开发，程序的共享性也是非常重要的。Windows 2000 的开发人员有 3000 多人，如果每一个人都自己定义字符串、链表等数据结构，那么开发效率就会降低，Windows 2000 恐怕到今天也不能出台。我所说的“共享”不是指将别人的代码复制到自己的代码中，而是指直接调用别人的代码，拿来即可用。这一方面可以减少代码的冗余性，另一方面可以增强代码的可维护性。如果别人的代码里有 Bug，只需修改他的代码，而调用此代码的程序不用进行任何修改就可以达到同步。



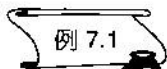
(6) 可测试性 (Testable)

在微软的产品开发里，除了软件开发人员，还有一部分工程师负责软件测试。软件测试人员会将开发代码拿来，一行一行地运行，看程序运行是否有错。如果软件开发人员的代码不可测试，那测试工程师就没有办法进行工作。因此可测试性在大型软件开发里是很重要的一点。

(7) 可移植性 (Portable)

可移植性是指程序写出来以后，不仅在 Windows 2000 里可以运行，在 NT/9X 下也可以运行，而且在 Linux 甚至 Macintosh 等系统下都可以运行。

所有这些特性都是一流代码所具备的特性。但是其中有些特性是会有冲突的。比如高效性，程序写的效率很高，就可能变得很复杂，牺牲的就是简洁。好的代码要在这些特性中取得平衡。



例 7.1

例 7.1 具体说明了一般代码可能存在的问题。该程序的源代码如下所示。

```
void MyGirlFriendFunc ( CORP_DATA InputRec, int
    CrntQtr, EMP_DATA EmpRec, float EstimRevenue, float
    YTDRevenue, int ScreenX, int ScreenY, COLOR_TYPE
    newColor, COLOR_TYPE PrevColor, STATUS_TYPE
    status, int ExpenseType)
{ int i;
  for (i=1; i<100; i++) InputRec.revenue[i] = 0;
    for (i=1; i<100; i++)
      InputRec.expense[i]= CorpExpense[i];
```

```

UpdateCorpDatabase (EmpRec) ;
EstimRevenue =YTDRevenue*4.0/ (float) CrntQtr;

NewColor = PreColor;
Status = Success;

if (ExpenseType==1) for (i=1;i<12;i++) Profit[i]
= Revenue[i]-Expense.Type1[i];
else if (ExpenseType == 2) Profit[i] = Revenue[i]
- Expense.Type2[i];
else if (ExpenseType==3) Profit[i] =Revenue[i]
-Expense.Type3[i];
}

```

这段程序初看写得不错，但实际上问题很多。

- (1) 可维护性低——它的程序名很抽象，而且程序中没有注释。
- (2) 简洁性不好——程序布局不好，代码应该对齐。
- (3) 稳定性差——如果变量 CrntQtr 等于零，程序会出现除零的错误，使得程序崩溃。
- (4) 共享性不好——程序没有一个统一明确的目的。
- (5) 可测试性差——着重标记的代码段不具备可测试性，如果 ExpenseType 等于 1，则程序测试起来非常困难，改正的方法就是将着重标记的代码段分行显示。
- (6) 性能低——for 循环的滥用。
- (7) 程序的参数太多，有些参数并没有使用，而且对参数的作用没有注释说明。
- (8) 常数（如 100, 4.0）用得不好，应该使用宏来定义。





7.2 编写代码的十大秘诀

根据我的工作经验，总结了十个写好代码的方法，希望对大家有帮助。

(1) 百家之长归我所用 (Follow Basic Coding Style)

其实写代码的方式有很多，每个人都有自己的风格，但是众多的风格中总有一些共性的、基本的写代码的风格，如为程序写注释、代码对齐，等等。

(2) 取个好名字 (Use Naming Conventions)

取个好的函数名、变量名，最好按照一定的规则起名。

(3) 凌波微步，未必摔跤 (Evil goto's? Maybe Not...)

这里我用“凌波微步”来形容 goto 语句。通常，goto 语句使程序跳来跳去，不容易读，而且不能优化，但是在某种情况下，goto 语句反而可以增强程序的可读性。

(4) 先发制人，后发制于人 (Practice Defensive Coding)

Defensive Coding 指一些可能会出错的情况，如变量的初始化等，要考虑到出现错误情况下的处理策略。

(5) 见招拆招，滴水不漏 (Handle The Error Cases: They Will Occur!)

这里的 Error Case (错误情况)，是指那些不易重现的错误。如果不对 Error Case 进行处理，程序在多数情况下不会出错，但是一旦出现异常，程序就会崩溃。





（6）熟习剑法刀术，所向无敌（Learn Win32 API Seriously）

我用“剑法刀术”来形容 Win32 API。Win32 API 经过了很多优秀开发人员的不断开发、测试，其效率很高，而且简洁易懂，希望大家能掌握它，熟悉它，使用它。

（7）双手互搏，无坚不摧（Test, but don't stop there）

这里的测试不是指别人来测试你的代码，而是指自己去测试。因为你是写代码的原作者，对代码的了解最深，别人不可能比你更了解，所以你自己测试时，可以很好地去测试那些边界条件，以及一些意想不到的情况。

（8）活用断言（Use, don't abuse, assertions）

断言（assertion）是个很好的调试工具和方法，希望大家能多用断言，但是并不是所有的情况下都可以用到断言。有些情况使用断言反而不合适。

（9）草木皆兵，不可大意（Avoid Assumptions）

是指在写代码时，要小心一些输入的情况，比如输入文件、TCP 的 sockets、函数的参数等等，不要认为使用我们的 API 的用户都知道什么是正确的、什么是错的，也就是说一定要考虑到对外接口的出错处理问题。

（10）最高境界，无招胜有招（Stop writing so much code）

意思就是说尽量避免写太多的代码，写得越多，出错的机会也越多。最好能重用别人开放的接口函数或直接调用别人的 API。

下面分别讲述这十个方法。所举的例子都是我在研究院和微软总部做开发时遇到的一些代码。





7.2.1 百家之长归我所用 (Follow Basic Coding Style)

当你坐下来写代码时，首先想到的一件事就是：这段代码不是为我写的，将来看这段代码的很可能不是写代码的本人，因为代码的存在有可能比人的记忆力所能记住的时间要长。在写代码的时候要想到，你写的这段代码别人可能会修改它。

不知道大家是否有这方面的经验，我自己就经历过好几次。有一天，一位同事拿着一段代码跑过来找我：“林斌，你这段代码是怎么跑的？我怎么看不懂？”

我把那段代码拿过来一看，第一个反应就是：“这代码不是我写的。写得这么乱，又糟糕，还没有注释，根本看不懂！”

我的同事说：“肯定是你写的！我在检测文件里看见你的名字了，你还说明了这段代码的功能。但是我就是看不懂它是怎么运行的！”

于是我把那段代码仔细看了看，又想了想，觉得有点像我以前写的。然后我们坐下来一起分析，看了半天，最后还是他看懂了，告诉我代码是怎么运行的，反过来给我讲了一通。我只有连连说道：“对！对！对！我当时就是这么想的！”

这样的情形在我身上发生过很多次，令我非常的尴尬。明明代码是自己写的，但就是想不起来它是干什么的！

所以我们在写代码的时候，一定要写基本的注释。有很多地方可以加注释，如文件头、函数头、文档之间一些重要的数据结构前，等等。

其他一些好的代码风格包括以下方面。

- ◀ 给函数和变量取个好名字；
- ◀ 代码要对齐，将 Tab/Indent 设置为四个字符；
- ◀ 少用参数；
- ◀ 尽量少用宏，写宏时用带下划线的大写字母。例如：
`#define MY_MACRO_NUMBER 100;`
- ◀ 在函数的开始处定义所有的变量；



- ◀ 限制函数的长度;
- ◀ 避免在程序中使用常量 (constant), 应该用宏或枚举代替常量。例如: `#define MY_MAX_LENGTH 1024;`
- ◀ 即使是单行的程序也用括号括起来。例如:

```
If (m_fInitialized) {
    hr = S_OK;
}
```

上述就是一些比较好的风格, 这些特点将会使代码看起来非常整洁、亲切。

例 7.2

下面这段代码是从以前 Windows 代码中抽出的。

```
DWORD Status = NO_ERROR;
LPWSTR ptr;
Status = f( ..., &ptr );
if ( Status isnot NO_ERROR or ptr is NULL )
    goto cleanup;
```

上述代码中的 `is`, `isnot` 和 `or` 让人不知所云, 它既不是新的 C/C++ 的关键字 (keywords), 也不是重载的操作符, 只是是一些会引起歧义的宏。这些宏可能是在该函数的前面定义的, 或者是在该函数所在文件的前面定义的, 甚至是在另外一个头文件中定义的。这样的事情大家千万别做, 不要自己发明编程语言, 而应该直接使用现在已存在的语言结构, 这样大家都能理解。为了避免程序将来可能出现的错误, 还要注意将条件加上括号, 以及对变量进行初始化。

我们看一下改正的版本。

```
DWORD    Status = NO_ERROR;
LPWSTR   ptr = NULL;
Status = f( ..., &ptr );
if ( (Status != NO_ERROR) || (ptr == NULL))
    goto cleanup;
```

前面我们提到一个程序不要写得太长，那么一个函数到底多长比较合适呢？在以前的 Exchange Server 中执行次数很多的一个函数超过了 1400 行！这么长的函数用在实际中是非常不好的，它极易出错，而且出错后很难修改。

当然，函数的长度也没有一个固定的标准，规定函数一定不要超过多少行是不切合实际的。但是当函数超过 200 行时，就要特别注意了。1986 年，有人对 IBM 的 OS/360 进行过统计，最易出错的函数是超过 500 行的函数。1991 年，有人对 148 000 行函数进行过统计，少于 143 行的函数的出错率比长函数的出错率要低 2.4 倍。

例 7.3

```
int DPCompare ( char *str1, char *str2 , int *diff ,
               int *len, int *sub, int *ins, int *del, int
               *EnglishNum = NULL, int *ErrorEnglishNum = NULL)
{
    int *R;
    int *B;
    int n,m;
    int i,j,k;
    int r1,r2,r3;
```

```

struct CHARBLOCK *char_block1, *char_block2;
struct COMPARE *res;

// New Space
char_block1 = new CHARBLOCK[strlen (str1) +3];
char_block2 = new CHARBLOCK[strlen (str2) +3];

// Initialization
R[0*m+0] = 0;
B[0*m+0] = 0;

// Iteration
for (k=1;k<=n+m;k++) {
    for (i=0;i<=__min (k,n);i++) {
        .....}
    }
}

```

这段代码非常糟糕：变量名都是些 $l, j, k, r1, r2, r3$ 之类的东西，根本不能从中看出变量的用途，而且程序中的注释也根本没有起到注释的作用。

7.2.2 取个好名字 (Use Naming Conventions)

匈牙利标记法 (Hungarian Notation) 是现在应用最广泛的一套命名规则，它的规则很简单：

[Prefix]-BaseTag-Name

其中 [Prefix] 是可选的，BaseTag 是数据类型的缩写，Name 就是



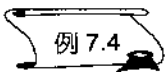
其名字。

标准的[Prefix]包括:

- p – 代表指针, 如 CHAR* psz;
- rg – 代表集合, 如 DWORD rgType[...]
- c – 代表计数器, 如书本的个数 DWORD cBook;
- h – 代表处理, 如 HANDLE hFile;

标准的“基本标记 (BaseTag)”

void	-> v	int	-> i
BOOL	-> f	UINT	-> ui
BYTE	-> b	CHAR	-> ch
WCHAR	-> wch	ULONG	-> ul
LONG	-> l	DWORD	-> dw
HRESULT	-> hr	fn	-> function
sz	-> NULL str	USHORT, SHORT, WORD	-> w
C++ 成员变量由 m 开头: m_			
全局变量以 g 开头: g_			



(1) 指明 C++对象是否被初始化的标记:

```
BOOL m_fInitialized;
```

(2) 会话的 ID:

```
DWORD dwSessionID;
```

(3) C++对象的 ref 计数器:

```
LONG m_cRef; // 省略了 'l'
```





(4) 指向 BYTE 缓冲区的指针:

```
BYTE* pbBuffer;
```

(5) 全局日志文件名的缓冲区:

```
CHAR g_szLogFile[MAX_PATH];
```

(6) 指向全局日志文件的指针:

```
CHAR* g_pszLogFile;
```

7.2.3 凌波微步，未必摔跤 (Evil goto's? Maybe Not...)

我用“凌波微步”来形容 goto 语句。很多写 C 语言的程序员都对 goto 语句深恶痛绝，因为 goto 语句会产生不清晰、不易读的代码，另外 goto 语句会使得编译器很难去优化程序。如果没有 goto 语句，编译器可以将发行版本进行很好的优化。如果我们查看优化后的机器码，会发现编译器将程序优化成非常简洁、高效的机器码。

但是 goto 语句是有两面性的。在某些情况下，goto 语句可以减少重复的代码，增强程序的可读性。例如，在单一入口、单一出口的情况下，用 goto 语句的效果是非常好的。

例 7.5

本例子中 goto 语句导致程序非常难读。

```
START_LOOP:
If (fStatusOk) {
    if (fDataAvaialbe) {
        i = 10;
        goto MID_LOOP;
    } else {
```

```
        goto END_LOOP;
    }
} else {
    for (I = 0; I < 100; I++) {
MID_LOOP:
        // lots of code here
        --
    }
    goto START_LOOP;
}
END_LOOP:
```

在这段代码中，由于 `goto` 语句的存在，程序跳来跳去，从 `if` 中跳出，又跳回到 `if` 中，让人找不到逻辑性，这是非常糟糕的。

但是 `goto` 语句也有好的情况，请看下面的代码。

例 7.6

```
HRESULT Init ()
{
    pszMyName = (CHAR*) malloc (256);
    if (pszMyName == NULL) {
        return hr;
    }
    pszHerName = (CHAR*) malloc (256);
    if (pszHerName == NULL) {
        free (pszMyName);
        return hr;
    }
}
```



```

}

pszHisName = (CHAR*) malloc (256);
if (pszHisName == NULL) {
    free (pszMyName);
    free (pszHerName);
    return hr;
}
... ..
free (pszMyName);
free (pszHerName);
free (pszHisName);
return hr;
}

```

这段代码中并没有使用 goto 语句，但是代码中重复的语句非常多，而且这段代码的维护性极差。如果再在其中加一条分配内存的语句，就要在三个位置上分别添加相应的 free 语句，要是漏了一处，就将导致程序的错误。如下面的代码所示。

```

HRESULT Init ()
{
    pszMyName = (CHAR*) malloc (256);
    if (pszMyName == NULL) {
        return hr;
    }
    pszHerName = (CHAR*) malloc (256);
    if (pszHerName == NULL) {
        free (pszMyName);
    }
}

```

```
    return hr;
}

pszItsName = (CHAR*) malloc (256);
if (pszItsName == NULL) {
    free (pszMyName);
    free (pszHerName);
    return hr;
}

pszHisName = (CHAR*) malloc (256);
if (pszHisName == NULL) {
    free (pszMyName);
    free (pszHerName);
    free (pszItsName);
    return hr;
}

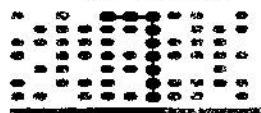
... ..

free (pszMyName);
free (pszHerName);
free (pszHisName);
free (pszItsName);
return hr;
}
```

下面再看一下使用 goto 语句的情况。在这里，程序有单一的入口与出口，且没有重复的代码。

```
HRESULT Init ()
{
```

第7章



10 Things You Can Do to
Write Better Code
写好代码的十个秘诀

```
pszMyName = (CHAR*) malloc (1);
if (pszMyName == NULL)
    goto Exit;
pszHeName = (CHAR*) malloc (1);
if (pszHeName == NULL)
    goto Exit;
pszHiName = (CHAR*) malloc (1);
if (pszHiName == NULL)
    goto Exit;
-
Exit:
    if (pszMyName)
        free (pszMyName);
    if (pszHeName)
        free (pszHeName);
    if (pszHiName)
        free (pszHiName);

    return hr;
}
```

这段代码很容易维护。如果要再添加一条分配内存的语句，只需在 `exit` 块中加一条 `if...free` 语句即可。如下面的代码所示。

```
HRESULT Init ()
{
    pszMyName = (CHAR*) malloc (1);
    if (pszMyName == NULL)
        goto Exit;
```



```
pszHeName = (CHAR*) malloc (..) ;
if (pszHeName == NULL)
    goto Exit;
pszItName = (CHAR*) malloc (..) ;
if (pszItName == NULL)
    goto Exit;
pszHiName = (CHAR*) malloc (..) ;
if (pszHiName == NULL)
    goto Exit;

...
Exit:
if (pszMyName)
    free (pszMyName) ;
if (pszHeName)
    free (pszHeName) ;
if (pszItName)
    free (pszItName) ;
if (pszHiName)
    free (pszHiName) ;

return hr;
}
```

所以，使用 goto 语句要遵循以下原则：

- ❏ 在程序为单一入口、单一出口时可以使用 goto 语句。
- ❏ 使用 goto 语句时千万不要往回跳，可以向前跳。
- ❏ 尽量不要用多于一个的 goto 语句标记，这会导致程序跳来跳去，不容易读。

- ✎ 要确保在使用 goto 语句后不要产生程序不可能执行到的一些代码。

7.2.4 先发制人，后发制于人（Practice Defensive Coding）

Defensive coding 是指一些可能会出错的情况，例如：

- ✎ 要注意变量的初始化问题。在大多数情况下，变量不必初始化，并且不会有问题。但是，如果在定义变量时没有对变量进行初始化，而是在中间将其赋值，则可能会出现下述情况：在赋值之前使用了一段代码去判断该变量的值；或者在赋值之前使用了 goto 语句跳过了该赋值语句，而最后要释放该变量所占内存 memory。由于该变量还没有被赋值，它的值是随机的，因此程序无法判断该变量的值，同样也不能释放它所占的内存，从而导致程序出错。
- ✎ 一定要校验函数的返回值。我们不能认为调用一个函数总会成功，要考虑到如果调用失败，应该如何处理。
- ✎ 要保持代码的简洁。通常，一段简洁的代码，其性能也会很好。
- ✎ 测试时要多运行几个线程。有些程序在一个线程下运行是正常的，但是在多个线程并行运行时就会出现問題；而有些程序在一个 CPU 下运行几个线程是正常的，但是在多个 CPU 下运行时就会出现問題，因为单 CPU 运行线程只是狭义的并行，多 CPU 一起运行程序，才是真正的并行运算。
- ✎ 要注意避免计算上的错误，比如除零错、符号的转换错误等。

在下面的例子中，变量 hr 没有初始化。如果 LocalAlloc 函数调用失败，则 hr 不会被赋值。因此，该段代码返回的值为随机值。这个随机值很可能等于 S_OK，进而造成调用者的错觉，认为调用成功。





例 7.7

```
HRESULT hr;
LPSTR str = (LPSTR) LocalAlloc (LPTR, size);
if (str) {
    ... // process appropriately
    hr = S_OK;
}
return hr;
```

如果做这样的改正，将 hr 初始化：

```
HRESULT hr = S_OK;
LPSTR str = (LPSTR) LocalAlloc (LPTR, size);
if (str) {
    ... // process appropriately
    hr = S_OK;
}
return hr;
```

做了这样的改正后，即使分配内存没有成功，程序也会返回 S_OK。

其实要改正上述问题很简单，只需加一个判断条件即可：如果 LocalAlloc 函数调用成功，则返回 S_OK，否则返回 E_OUTOFMEMORY。如下面的代码所示。

```
HRESULT hr = S_OK;
LPSTR str = (LPSTR) LocalAlloc (LPTR, size);
if (str) {
    ... // process appropriately
```



```

    hr = S_OK;
} else {
    hr = E_OUTOFMEMORY;
}
return hr;

```

下面的例 7.8 是很多人容易犯的错误。

例 7.8

```

BOOL SomeProblem (USHORT x)
{
    while (--x >= 0) {
        [-]
    }
    return TRUE;
}

```

这个程序会导致死循环，因为无符号数（USHORT）永远也不会为负数，while 将永远执行下去。

例 7.9

```

wcsncpy (wcServerIpAddress, WinsAnsiToUnicode (cAddr,
    NULL));

```

注意：WinsAnsiToUnicode 是一个 API，它将所分配的 ANSI 串转换为新的 UNICODE 串。



这段代码在任何情况下都会出现问题：当 `WinsAnsiToUnicode` 失败返回 `NULL` 时，会导致程序的崩溃，因为 `wcscpy` 没有对 `NULL` 的情况进行处理；当 `WinsAnsiToUnicode` 返回所分配的内存空间时，由于 `wcscpy` 没有将所分配内存收回，会导致内存占用越来越多，最终会使程序无法运行。

只需加一个中间变量，就可以将它改正。如果 `WinsAnsiToUnicode` 成功，就使用获得的内存，用完之后释放掉；如果 `WinsAnsiToUnicode` 失败，就直接返回一个错误码。

```
LPWSTR tmp;

tmp = WinsAnsiToUnicode (cAddr, NULL);

if (tmp != NULL) {
    wcscpy (wcServerIpAddr, tmp);
    WinsFreeMemory ((PVOID) tmp);
} else {
    return (ERROR_NOT_ENOUGH_MEMORY);
}
```

7.2.5 见招拆招，滴水不漏（Handle The Error Cases: They Will Occur!）

错误情况 (Error Case)，是指那些不易重现的错误。一定要对 Error Case 进行处理，免得程序崩溃。其中最常出现的情况如下。

- ◀ 内存耗尽。不要认为 100 B 内存很小，不会出错，因为在系统运行时，尽管我们自己的程序申请的内存很少，但是不能保证别人的程序申请的内存也很少。在开发过程中就经常出现这样的情况：别人的程序申请的内存太多，导致我的程序由于内存申请不到而崩溃了。
- ◀ 异常。在 C++ 中经常会出现异常，要做好处理它的准备。用

过 MFC 的用户都知道, 如果出现异常, 就必须处理它, 否则程序随时会崩溃。

- ✧ 网络中断。不要以为发送一个 socket 都会成功, 尤其是如果用异步 socket 做一些网络软件, 客户端将数据包发送到服务器端时很容易出错。由于是异步, 所以尽管发送方是正确的, 但过一段时间接收方就可能出错, 而且这种异步错误是很难检测的。有些错误可以在调试过程中借助一些手段产生出来, 有些错误则是不能重现的。在错误不能重现的情况下, 我们就要做一些工具, 迫使错误情况出现。

另外, 在编程的过程中还应该注意:

- ✧ 在 C++ 对象的构造函数中不要做一些可能会失败的操作, 如内存的操作, 因为在构造函数中出错后是没有方法知道的。
- ✧ 处理错误情况时, 要释放分配到的资源; 接口中应该清楚地定义程序的行为, 如返回状态, 异常的处理等。要让调用者清楚地知道接口的定义。
- ✧ 千万不要忽略错误, 从而造成程序崩溃或退出。

例 7.10

```
CWInfFile::CWInfFile () {
    m_plLines      = new TPtrList (); // ...
    m_plSections   = new TPtrList (); // ...
    m_ReadContext.posLine = m_plLines->end ();
    ...
}
```

这段程序中最大的问题是 MFC 的 new 操作, 如果失败了, 它们会产生异常。如果前一个 new 操作分配正常, 而后面的 new 分配出错, 则前一个变量的分配会导致内存的泄漏。解决的办法是用一个

try...catch 语句来处理：如果出现异常，则删除变量，并释放内存。

```
CWInfFile::CWInfFile () {  
    try {  
        m_plLines      = new TPtrList () ; // ...  
        m_plSections   = new TPtrList () ; // ...  
        m_ReadContext.posLine = m_plLines->end () ;  
        . . .  
    } catch ( . . . ) {  
        if (m_plLines) delete m_plLines;  
        if (m_plSections) delete m_plSections;  
    }  
}
```

但是，这里面还有一个问题：由于构造函数是最先执行的，如果 new 分配出错，就会执行 delete 语句，而此时变量 m_plines 和 m_plsections 还没有初始化，若对它们进行 delete 操作，就会出错，所以我们应该在构造函数中将其初始化。

```
CWInfFile::CWInfFile ( ) : m_plLines ( NULL ) ,  
    m_plSections (NULL) {  
    try {  
        m_plLines      = new TPtrList () ; // ...  
        m_plSections   = new TPtrList () ; // ...  
        m_ReadContext.posLine = m_plLines->end () ;  
        . . .  
    } catch ( . . . ) {  
        if (m_plLines) delete m_plLines;  
        if (m_plSections) delete m_plSections;  
    }  
}
```

请注意，这里如果用的不是 MFC，则 new 操作出错时就会返回 NULL，而不会抛出异常，因此第三个变量的初始化就会出错。

```

CWInfFile::CWInfFile ( )      : m_plLines ( NULL ) ,
    m_plSections (NULL) {

try {

    m_plLines      = new TPtrList ( ) ; // ...
    m_plSections   = new TPtrList ( ) ; // ...
    m_ReadContext.posLine = m_plLines->end ( ) ;

    . . .

} catch ( . . . ) {

    if (m_plLines) delete m_plLines;

    if (m_plSections) delete m_plSections;

}

}

```

下面的例 7.11 讲的是在构造函数中不要使用一些会失败的操作。

例 7.11

```
Class foo {
private:
    CHAR* m_pszName;
    DWORD m_cbName;
```




```
public:
    foo (CHAR* pszName) ;
    CHAR* GetName ()
    {return m_pszName;}
...
};

foo::foo (CHAR* pszName)
{
    m_pszName = (BYTE*) malloc (NAME_LEN) ;
    if (m_pszName == NULL) {
        return;
    }
    strcpy (m_pszName, pszName) ;
    m_cbName = strlen (pszName) ;
}
.....

foo* pfoo = new foo ("MyName") ;
if (pfoo) {
    CHAR c = * (pfoo->GetName ()) ;
}
```

由于在构造函数中无法返回错误码，当我们使用 `new` 操作符创建一个 `foo` 对象时，会调用其构造函数，如果 `malloc` 函数出错，构造函数就直接返回，则 `pfoo` 就没有分配值，这样在后面的程序中对 `pfoo` 的访问就会出错。因此，在构造函数中不要用一些会失败的操作。

在下面这个改动后的版本中，构造函数中不再包含失败的成分，



而是增加了一个 `init` 函数来完成内存的分配操作。在使用中先调用 `new` 函数再调用 `init` 函数，检验返回值，判断是否执行正确，有错就将其返回。

```
Class foo {
private:
    CHAR* m_pszName;
    DWORD m_cbName;

public:
    foo();

    HRESULT
    Init (CHAR* pszName);

...
};

foo::foo()
{
    m_cbName = 0;
    m_pszName = NULL;
}

HRESULT foo::Init (CHAR* pszName)
{
    HRESULT hr = S_OK;
    if (pszName) {
        m_cbName = lstrlen (pszName);
        m_pszName = (CHAR*) malloc (m_cbName+1);
        if (m_pszName == NULL) {
```

```
    hr = E_OUTOFMEMORY;
    return hr;
}

strcpy(m_pszName, pszName);
} else {
    hr = E_INVALIDARG;
}
return hr;
}
```

7.2.6 熟练剑法刀术，所向无敌 (Learn Win32 API Seriously)

如果做 Windows 的开发，我认为 Win32 API 是一个很有用的工具。它经过了很多优秀的开发人员花了大量的时间去开发、测试、完善、优化，其性能非常高，而且代码简洁，容易读懂，希望大家能花时间去掌握它，熟悉它，使用它。可以从 MSDN 上的 SDK 文档中读到这些 API，其中有很多很好的例子，介绍了这些 API 什么时候会出错，出错了该如何处理，等等。

在下面的例 7.12 中，变量的初始化、移动等操作都是利用循环做的。

例 7.12

```
for ( i=0; i< 256; i++) {
    re[i] = 0;
    im[i] = 0;
}
for ( k = 0; k < 128; k++)
```

```

    AvrSpec[k] = 0;
-- --
#define FrameLen 200
for (k=0; k<5*40*FrameLen; k++)
    lsp[k] = lsp[k + 40*FrameLen];
-- --
for (k = 0; k < FrameLen; k++) {
    audio[k] = vect[k];
}
--

```

其实前面三个变量的初始化全部可以用 ZeroMemory 这个 API 函数来实现，后面两个变量的移动和赋值操作可以分别用 MoveMemory 和 CopyMemory 这两个 API 函数来实现，如图 7.1 所示。

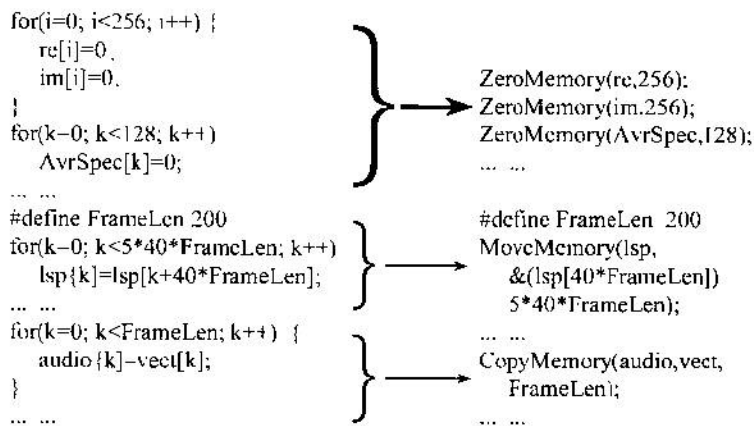


图 7.1 用 API 函数代替循环来实现变量初始化、移动和赋值操作

如果将上面的左右两段代码编译好后，分别打开它们的机器码，就会发现 ZeroMemory 函数的效率非常高，它是用 Intel 的 Movememory 函数来实现的，只需要几个循环，而 For 循环却需要很

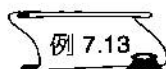


长的代码才能完成同样的功能。



注：MoveMemory 与 CopyMemory 的区别是：MoveMemory 考虑了重叠的问题，而 CopyMemory 不考虑重叠的问题。

下面的例 7.13 讲述对函数返回值的错误处理问题。



```
TCHAR WindowsDir[MAX_PATH];  
TCHAR DriveLetter;  
GetWindowsDirectory (WindowsDir, MAX_PATH);  
DriveLetter = WindowsDir[0];  
...
```



注：GetWindowsDirectory 返回的是 Windows 的主目录。

上述程序遗漏了检查 GetWindowsDirectory 函数调用的返回值，但是实际上大部分开发人员都会这样写这段代码。在我们的印象中，GetWindowsDirectory 函数是不会出错的，因为它访问的是系统的 Path 环境变量，而此变量在系统启动时就被设置好了，在任何 Windows 系统的内存中都存在，所以 GetWindowsDirectory 函数不可能出错。

但是 MSDN 说它会失败，而我在写代码时从未遇到调用失败的情况，也没有见到别人调用失败，于是我就认为 GetWindowsDirectory 函数不会失败，这其实是一种固定思维在作怪。

实际上，有一种情况下 GetWindowsDirectory 函数是会失败的：终端服务器。因为客户端是取服务器端的内容，它要将 GetWindowsDirectory 中的内容存入内存，因此要分配内存空间，但

是在内存分配时就有可能出错。如果出错，就会返回错误。但是开发人员并没有预见到错误的出现，没有处理错误，这样就会在终端服务器上产生灾难性的结果。如果对 API 很熟悉，在这些细节上就不会犯错误，所以一定要熟悉 API。

改正方法很简单，加一个 if...else 来判断。如果出错，返回错误码。

```
TCHAR WindowsDir[MAX_PATH];
TCHAR DriveLetter;
if (!GetWindowsDirectory(WindowsDir, MAX_PATH)) {
    return E_OUTOFMEMORY; // proper recovery
}
DriveLetter = WindowsDir[0];
...
```

7.2.7 双手互搏，无坚不摧（Test, but don't stop there）

本书第 9 章和第 10 章对与测试有关的工作进行了深入的分析。本部分讲述作为一个软件开发人员，如何测试自己的程序，使代码做得更好，更加稳定。就我个人的经验来说，如果没有测试过代码，它就不可能正确运行。

另外，在同一组的开发人员之间做得很多的一件事就是：别人来对你的代码进行检查，反过来你对别人的代码进行检查。在这个过程中，不仅是希望检查的人来发现你的代码中的问题，或是你去发现别人代码中的问题，更重要的是在向别人讲解你的代码时，可以发现自己遗漏的地方和问题，理顺自己思路。



例 7.14

```
HRESULT hr = NOERROR;

// Make sure the arguments passed are valid
if (NULL != m_paConnections) {
    ... // do the right thing
}
else hr = S_FALSE;
if (SUCCEEDED(hr)) {
    if (NULL != m_paConnections[0].pUnk)
        ...
}
```

这段程序的错误是：C 的 `SUCCEEDED(S_FALSE)` 返回的是 `TRUE`。如果 `m_paConnection` 是 `NULL`，则程序就会崩溃。在出错时将 `hr` 设置为 `E_INVALIDARG` 就可以了。

```
HRESULT hr = NOERROR;

// Make sure the arguments passed are valid
if (NULL != m_paConnections) {
    ... // do the right thing
}
else hr = E_INVALIDARG;
if (SUCCEEDED(hr)) {
    if (NULL != m_paConnections[0].pUnk)
        ...
}
```

下面这段程序是我在开发 `Exchange Server` 时写的一段代码，当时写完以后我没有测试它，因为这段代码实在是太简单了，只有几

行代码：取文件的长度，如果出错就返回。于是我仅仅是编译通过后就将其提交（checkin）到实际产品中了。

结果第二天早上当我到办公室的时候，发现我的三位上司都已经铁青着脸在那里等我了。原来，整个 Exchange Server 都运行不起来了！因为我的这段代码被加在了 Exchange Server 启动代码序列中，当 Server 启动时，由于我这段代码的错误，一启动就失败，导致了 DOA（Dead On Arrival）。

例 7.15

```
//
// Get file size first
//
DWORD dwFileSize = GetFileSize( hFile, NULL );
if ( dwFileSize = -1 ) {
    // what can we do ? keep silent
    ErrorTrace ( 0, "GetFileSize failed with %d",
        GetLastError () );
    return;
}
```

注：GetFileSize 调用失败时将返回-1。

这段代码的错误在了：if 的判断条件写成了赋值，所以无论怎样都会出错，然后返回。

其实改进的方法很简单：就是将-1 移到前面。这样，如果你遗漏了一个“=”，编译时编译器就会发现错误。所以在 if 语句中，要把常量放在前面。



注意


```
//  
// Get file size first  
//  
DWORD dwFileSize = GetFileSize( hFile, NULL );  
if ( -1 == dwFileSize ) {  
    // what can we do ? keep silent  
    ErrorTrace ( 0, "GetFileSize failed with %d",  
        GetLastError() );  
    return;  
}
```

这件事情给我的教训是很深刻的。

7.2.8 活用断言 (Use, don't abuse, Assertions)

断言是一个很好的工具，它在调试里很有用。可以用断言做助手，帮助开发人员发现问题。经常使用断言对开发人员固然很有帮助，但是并不是所有的情况下都可以使用断言。看下面的例 7.16。

例 7.16

```
CHAR * pch;  
pch = GlobalAlloc (10);  
ASSERT (pch != NULL);  
pch[0] = 0;
```

通常，断言在 free/realse 版本里不会做任何事情。当在 free/realse 版本里运行上述例子时，假如申请内存不成功，ASSERT 也不会做任何操作，而是直接执行后面的语句，程序就会出错了。所以不要在

可能出错的情况下使用断言。

改进方法是：加一个 if 判断语句，如果出错，则返回错误。

```
CHAR * pch;
pch = GlobalAlloc(10);
If (NULL == pch) {
    return E_OUTOFMEMORY;
}
pch[0] = 0;
```

7.2.9 草木皆兵，不可大意 (Avoid Assumptions)

当你开放一个接口或 API 供别人调用时，一定要检测输入的参数，包括数据、文件、socket 等各种各样的参数，因为这些数据很有可能会造成程序出错。使用者也许不会按你定义的标准接口进行输入，他可能会将各种各样的“垃圾”输入你的接口，所以你要保证你的系统在输入各种数据时，都可以运行起来，不会崩溃。

例 7.17

```
CIRRealExtractor::CIRRealExtractor ( HRESULT* hr,
    const BITMAPINFOHEADER *lpbmi, const void *lpvBits,
    const RECT *prectBlock)
{
    --
    *hr = ContainDIB (lpbmi, lpvBits, prectBlock);
};
```

在这里，如果输入的 hr 参数的值为 NULL，则 C++ 的构造函数

就会出问题。当然，调用此函数的用户通常不会输入 `NULL` 作为 `hr` 参数的值，但是万一出现了这种情况，一定要保证你的程序不会出错。改正的方法是，加一个 `if` 判断语句。

```
CIRRealExtractor::CIRRealExtractor ( HRESULT* hr,
    const BITMAPINFOHEADER *lpbmi, const void *lpvBits,
    const RECT *prectBlock)
{
    --
    if (hr) {
        *hr = ContainDIB (lpbmi, lpvBits, prectBlock);
    }
};
```

另外，我们在写一个函数时，可以使用 `assert` 语句来抓住不合法的情况。例如，可以使用 `assert` 语句来判断输入参数是否合法。

```
CIRRealExtractor::CIRRealExtractor ()
--
HRESULT CIRRealExtractor::Init (
    const BITMAPINFOHEADER *lpbmi, const void *lpvBits,
    const RECT *prectBlock)
{
    ---
    // lpbmi, lpvBits, and prectBlock can never be NULL
    ASSERT ( lpbmi != NULL);
    ASSERT ( lpvBits != NULL);
    ASSERT ( prectBlock != NULL);
```

```
return ContainDIB (lpbmi, lpyBits, prectBlock);
};
```

例 7.18

系统在运行了很久时间以后会无缘无故地重启，这种情况在 NT4.0 下经常会出现，其直接原因是它调用了 `services.exe`，这个进程的作用是启动 NT 里所有的 Servers。那么为什么在低内存的情况下会重启呢？原来它调用了一个名为 `wscicmp` 的 API 函数。如果查看其源代码，就会发现其中有这样一段代码：

```
---
if ( _locktable[locknum] == NULL ) {

    if ( (pcs =
        _malloc_crt (sizeof (CRITICAL_SECTION)))
        == NULL )
        _amsg_exit (_RT_LOCK);
}
---
```

它的作用是申请 `CRITICAL_SECTION` 内存空间，如果不成功，就退出，但它不是从函数中退出，而是使得调用它的进程整个退出。这是 C 运行库（C Runtime）的问题，其实 C 运行库中的很多 API 函数都有这个问题：申请 `CRITICAL_SECTION` 时，如果不成功，就将整个进程退出，最终导致系统的重启。因此大家在使用 C 运行库中的函数时要注意这一点。

第 7 章



好
代
码
的
上
个
秘
诀





7.2.10 最高境界，无招胜有招（Stop writing so much code）

写代码并不是写得越多越好，最好能尽量重用别人的代码，不要抄，而要直接调用他人写的 API。拷贝时更要小心，要知道拷贝代码的同时也是在拷贝 bug。

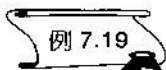
我对以前的 Windows 源代码曾做过一个大概的统计：其中有 100 多个不同的 new 全局操作符，9 个不同的 JPEG 实现，80 多个不同的字符串实现，拷贝了成千上万个一模一样的函数，3 个不同版本的 atlimpl.cpp，很多重复的 ANSI/UNICODE 的文件。

7.3 结束语

端正态度。

要为自己的代码自豪，但是不要觉得自己的代码是完美的，要不断地完善，不断地发现问题，寻找问题，然后改正问题。

要使自己写出的代码做到正确、可靠、简洁、高效（如速度快、内存占用率低等）不是一蹴而就的，这需要不断地总结自己在编写程序中出现的错误，不断地练习才能不断地提高。



例 7.19

```
Comment from PREFIX source code:  
  
/* In theory it might be possible to create the data  
   with the right scope, but my attempts to do so have  
   all created crashes of one sort or another. Since  
   we only use this information to improve the clarity  
   of our output, I'm just not creating it in this  
   situation. */
```



i.e., "I know the right thing to do but I couldn't make
it work so I gave up."

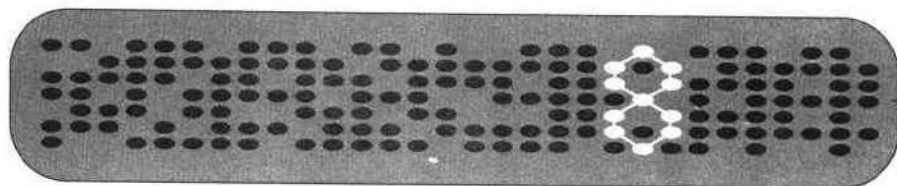
上面的注释是从一个实际的项目中抽出来的，是一个典型的态度不好的例子。

最后我要说的是：写程序时要使自己专注、认真，做到
快乐编程（Happy Coding）！

总结

第 7 章

10 Things You Can Do to
Write Better Code
写好代码的十个秘诀



第 8 章

如何提高程序的性能

Secrets of Software Performance

背景

如果你是一名软件开发人员，是不是经常因为程序的性能而受到老板的指责？你是不是经常因为程序的性能而被用户刁难？你是不是经常因为程序的性能而饱受等待之苦？那么，怎样才能尽量优化应用程序，提高其性能呢？本章，林斌根据自己多年在微软进行软件开发的实践经验，为我们提供非常棒的提高性能的方法，并列举了具体的案例学习，最后，还详细介绍了令无数软件开发人员头痛的内存问题。如果你是一名软件开发人员，阅读本章后，立即将这些方法应用到你的应用程序中，体验一下性能提高的喜悦吧。

林斌經理在做講座



本章内容概览

- ☞ 提高性能的方法
- ☞ 案例学习
- ☞ 内存

8.1 提高性能的方法

提高软件程序运行的性能有很多种方法，最常用的有以下几种。

(1) 使用速度更快的硬件

这是一个非常简单的方法，也是许多公司采用最多的提高软件效率的方法。显然，同样的一个程序，在微机、小型机和大型机上的运行速度是大不一样的。其实这种方法也相当于用钱来买程序的性能。花的钱越多，使用的计算机的速度越快，程序的性能自然就越高。

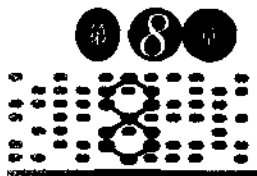
(2) 选择比较好的编程语言和编译器


例如，写同样一个程序，如果分别使用 C 语言和 BASIC 语言来完成，则使用 C 语言编写程序的效率和程序性能就要高一些；如果想使程序的运行效率更高，不妨使用汇编语言来编写；如果还觉得程序的性能不够，则可以选择一个比较好的编译器。

(3) 设计扩展性很强的应用程序

我们在进行软件设计时就必须考虑到如何提高程序的性能。这个方法主要是在程序开发之前进行的。例如：

- ◀ 根据硬件的特性来设计。例如 Intel X86 CPU 具有速度很快





的 L1 Cache 和 L2 Cache。如果计算机指令是在 Cache 中执行的，运行速度会比在 RAM 中快很多，而 RAM 的运行速度又比磁盘的运行速度快。我们在设计程序时应该考虑到硬件的这些特点，对程序进行灵活处理。

◀ 根据程序的要求选择正确的数据结构和算法。

(4) 对代码进行调整

这个方法主要在程序开发的末期进行。此时程序开发基本完成，运行也基本稳定下来，但在某些方面还可以进行调整。例如：


- ◀ 尽量不要调用那些速度慢、代价高的 Windows API。有些 Windows API（如完成读写文件，读写 Sockets 等操作的函数）是非常复杂、非常耗费资源的。因此要尽量避免调用它们。
- ◀ 如果一个小程序要被调用很多次，则应该采用各种方法尽量提高这个小程序的效率。

前两种方法的技术性都不是很强，因此不准备多讲。我主要讲后面两种方法，而重点又放在第（3）种方法。

8.2 案例学习

8.2.1 案例分析

下面看一个案例分析。这个案例是我来微软第二年做的一个设计，其要求如下：

- ◀ 设计一个可扩展的 SMTP 服务器；
 - ◀ 该服务器在具有一个 CPU、两个 CPU、4 个 CPU 甚至 8 个 CPU 的计算机上不但能够正确运行，而且性能随着 CPU 的增加而成倍增长；
 - ◀ 该服务器能够同时处理尽可能多的用户请求，并且能够较快
- 



地做出响应。

注：SMTP 是 Simple Mail Transport Protocol（简单邮件传输协议）的缩写，是一种传输电子邮件的标准协议。

在以上几点要求中，关键的一点是可扩展性。这在服务器的开发中是十分关键的。所谓“可扩展”，就是说服务器性能的提高与 CPU 数目的增加成正比。比如说，当只有一个 CPU 时，SMTP 服务器每秒能处理 40 封邮件；那么，如果是两个 CPU 的服务器，就应该能每秒处理近 80 封邮件；4 个 CPU 能处理 160 封，如此类推。实际上，考虑到 Context Switch 与别的额外开销，服务器性能要达到随 CPU 数目翻倍而成倍增长是不可能的。最理想的结果是使服务器性能最大限度地增加。

在开始动手实现这个案例之前，先来看一下 SMTP 服务器的实现原理。下面列出了一个被高度简化的 SMTP 服务器的框架实现代码：

```
// Read SMTP commands/data from sockets
If (ReadFile ( _ ))
{ // various housekeeping removed_ }
// Parse SMTP recipients and other headers
If (!ParseSMTPHeaders ( _ )) {
    // handle errors_
}
// Parse bodies
If (!ParseSMTPBodies ( _ )) {
    // handle errors_
}

// Local delivery or routing
If (LocalDelivery ( _ )) {
    Deliver ( _ );
}
```

```
} else {  
    Route ( ... );  
}  
  
// Send SMTP response through Socket  
If (WriteFile (...)) {  
    // various housekeeping skips...  
}
```

从上面的代码可以看出，SMTP 服务器的实现原理是非常简单的。程序首先会通过网络套接字（Sockets）读入一个用户请求（通过 ReadFile 函数实现）。这个用户请求中包括了 SMTP 邮件头、邮件体和一些 SMTP 命令。接下来，程序会对 SMTP 邮件头进行解析，以获取相关信息，如收件人的地址、发送时间和邮件主题（Subject）等。然后，程序再对 SMTP 邮件体进行解析，同样获取相关信息，如邮件正文的长度，是否含有附件，附件的格式等。做完解析过程以后，程序就要根据刚才所获取的信息进行判断，主要判断当前邮件是否为本地邮件。判断结束以后，使用 WriteFile 函数发送 SMTP 响应。WriteFile 函数会根据不同的判断结果做不同的事情。如果是本地邮件，则直接将其发送到本地邮箱（实际上为计算机上的一个子目录）即可；如果不是本地邮件，则需要通过 Socket 转发到下一个 SMTP 服务器，再由该 SMTP 服务器重复上面的处理过程。

由此，可以总结出 SMTP 服务器的实现过程，包括以下几部分：

- ◀ 读入用户请求；
- ◀ 解析邮件头；
- ◀ 解析邮件体；
- ◀ 判断是否为当前邮件；
- ◀ 发送邮件，根据不同情况进行本地分发或者路由转发。



当然，上述代码是完全抽象出来的。尽管 SMTP 服务器的实现原理非常简单，但实际的实现代码还是非常复杂的。

8.2.2 一个传统的设计

下面是一个传统的设计，其实现原理如图 8.1 所示。

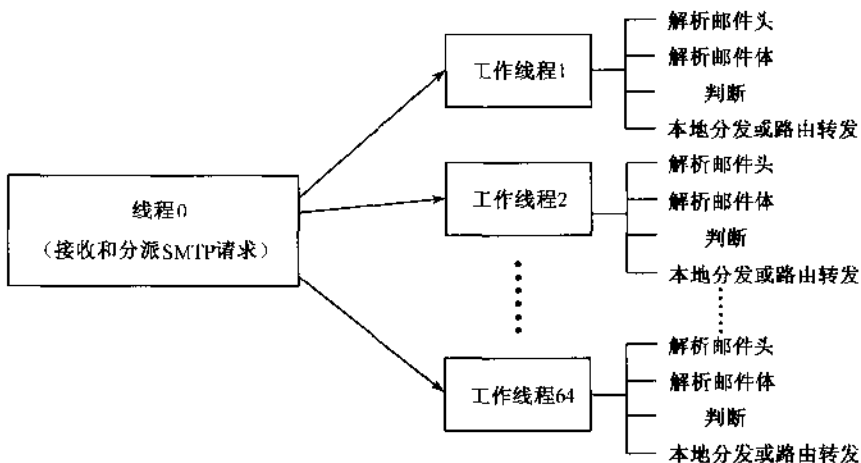


图 8.1 一个传统设计的示意图

这个设计非常简单。首先有一个线程用来不停地读入用户的 SMTP 请求，然后将这些请求分派给 64 个工作线程，这些工作线程则分别循环往复地依次完成以下工作：

- ◀ 解析 SMTP 邮件头；
- ◀ 解析 SMTP 邮件体；
- ◀ 判断是否为当前邮件；
- ◀ 发送邮件，根据不同情况进行本地分发或者路由转发。

这个设计有什么优点和不足呢？

首先来回顾一下硬件的发展过程，如图 8.2 所示。

从图 8.2 中可以看出，从 1992 年到 2000 年，CPU、RAM 和磁盘 3 种硬件都得到了高速发展。但从性能（速度）上来看，增长最快的





是 CPU，其次是 RAM，而磁盘的性能提高则很有限（尽管其容量已经有了很大的提高）。由此便造成它们之间的性能差距越拉越大。

为了减低三者的性能差距，Intel 在 x86 CPU 中增加了两级 Cache（缓存），即 L1 Cache 和 L2 Cache。L1 Cache 容量比较小，它只能存储 8KB 的指令和 8KB 的数据。但是它的速度非常快，实际执行一个指令时只需要大约 3 个时钟（clock）的时间。而 L2 Cache 要大一些，有 256KB，512KB，甚至 1MB，但是它的速度要稍微慢一些，通常执行一个指令需要大约 7 个时钟的时间。

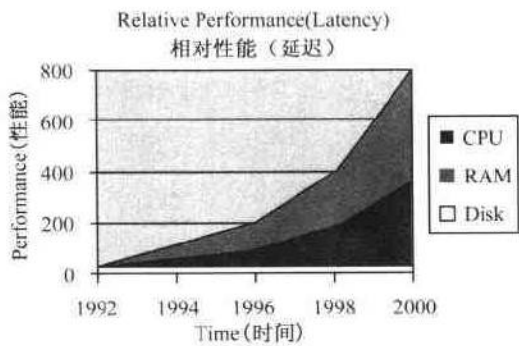


图 8.2 CPU, RAM 和 Disk 的发展过程

现在用一个比较形象的方法比较一下各种硬件的性能。假设要执行一条将寄存器置 0 的指令。如果这条指令和它所需要的数据 (0) 已经存储在 L1 Cache 中，假设 CPU 执行这个操作耗时 1s（为了说明方便，这里做了极大的夸张）的话，那么如果指令存储在 L2 Cache 中，CPU 执行这个操作将耗时 4~7s；存储在内存（RAM）中，CPU 执行操作将耗时 25~150s，而如果存储在硬盘中，CPU 执行同样一个操作将耗时 3 个星期！

由此可见，在做程序设计时，如果能够找到一个很好的设计方案，充分地利用 L1 Cache 或 L2 Cache，甚至 RAM，那么程序性能就会好。相反，如果所设计的程序需要大量地去读写硬盘，使操作系统不断地进行页面切换，程序的性能就会大打折扣。基于这一原





因，我们对上述的传统设计进行了修改。

8.2.3 一个更好的设计

这个新设计方案的实现原理如图 8.3 所示。

首先，也使用一个线程来不停地读入用户的 SMTP 请求，但是现在在每个 CPU 中只使用一个工作线程来处理这些请求。这个工作线程需要完成下面 4 个事件循环：

- ◀ 解析 SMTP 邮件头；
- ◀ 解析 SMTP 邮件体；
- ◀ 判断是否为当前邮件；
- ◀ 发送邮件，根据不同情况进行本地分发或者路由转发。

为每一步工作建立一个队列，每个队列可以有多个同样的工作。工作线程针对同一个队列重复完成处理所有的工作，才跳到下一个队列处理该队列中的工作。也就是说，工作线程首先要循环地解析在“SMTP 邮件头队列”中的所有 SMTP 邮件头，然后再循环解析所有在“SMTP 邮件体队列”中的 SMTP 邮件体；然后再处理下一个队列，如此类推，直到邮件被发送完，又回到第一个队列，如此不断循环。

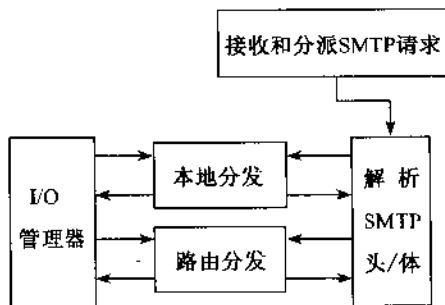


图 8.3 新设计的实现原理示意图

这个新设计的主要优点在于：它充分发挥了指令局部性和数据





局部性的特点。工作线程在处理每个队列时，由于队列中每一个操作所做的事情都是一样的（如不停地解析邮件头），因此各个操作的指令序列是完全相同的；而且，各个操作的数据也非常相近，都处在一个范围非常小的区域中。这样，在每一个循环中，经过一两次的操作，当指令和大部分的数据都被从磁盘上加载到 Cache 中之后，就可以重复调用，不再需要跑到磁盘上去调，从而极大地提高程序的性能。

那么，新设计的性能比传统设计的性能到底有多大提高呢？让我们来看一下它们的总体性能对比图，如图 8.4 所示。

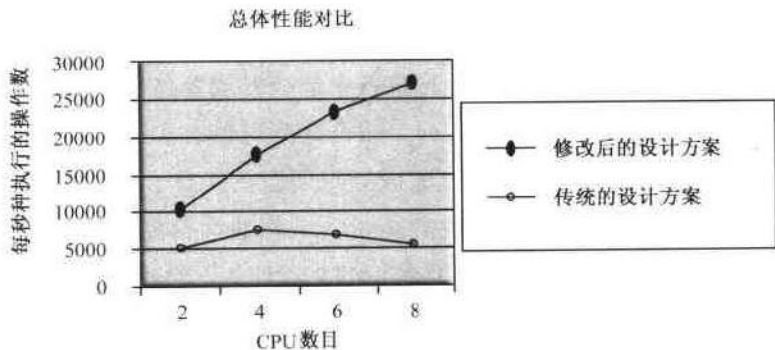


图 8.4 两种设计的性能对比图

图 8.4 是由我们的测试队伍得出来的。从中可以看出，在只有一个 CPU 的情况下，新设计的性能并不比传统设计的性能好多少。这是因为，尽管新的设计比传统设计要好，但是，新设计的程序也变得比传统设计复杂了一些，同时只有一个线程来处理 4 个队列，这些因素都会消耗程序的性能，因此就此消彼长了。但是，随着 CPU 数目的增多，新设计的性能优势就体现出来了：新设计的性能急剧上升，而传统设计的性能基本没有变化。也就是说，CPU 的多少对传统设计的性能提高起不到什么作用。在使用两个 CPU 时，新设计比传统设计快一倍；在使用 4 个 CPU 时，新设计比传统设计快两倍；而在使用 8 个 CPU 时，新设计比传统设计快四倍。





这一点对服务器是特别有用的。能够使用多个 CPU 并充分利用 CPU 及其所带的 Cache 的强大功能, 对 SMTP 服务器来说就更重要了。

8.2.4 总结

从上述两种设计方案的对比中, 可以归纳出以下结论:

- ◀ Cache 使用得正确与否会严重地影响到程序的整体性能和扩展性;
- ◀ 循环处理 (或批处理) 的使用可以显著地提高程序的性能;
- ◀ 要充分利用指令局部性和数据局部性的特点;
- ◀ 给每个用户分配一个线程的体系结构的效率很低, 并且扩展性不好。

8.3 内存

如果大家经常使用 Windows, 相信对 Windows 中的内存使用情况印象非常深。这里用到这样一个概念——Footprint (脚印), 它实际指的就是 Windows 中的内存使用情况。大家知道, Windows 对内存的需要是比较大的, 绝大多数应用程序在具有 16MB 内存的 Windows 98 系统或具有 32MB 内存的 Windows 2000 系统中都无法正常运行。为了提高程序的性能, 经常要做的一项工作就是减少程序的 Footprint。

8.3.1 Footprint 的概念

Footprint 是指一个应用程序的工作集。这个工作集是没有限制的, 它包括:

- ◀ 程序刚刚使用过的所有代码和数据;





- ◀ 程序动态分配的数据；
- ◀ 操作系统为支持应用程序所使用的资源。

这里的工作集实际上是由一些存储页面组成的，这些页面所使用的是真正的物理内存，而不是硬盘上的虚拟内存。

注：Windows 中有一个虚拟内存的概念，虚拟内存既可以保存在物理内存中，也可以保存在硬盘中。

8.3.2 虚拟内存的使用

在 Windows 2000 中，虚拟内存的大小可以达 4 GB，其中 2 GB 是给系统使用的，另外 2 GB 是给应用程序使用的。

一开始，虚拟内存是空的。后来随着程序的运行，代码和静态数据都会占用空间，然后程序调用的 DLLs 会占用相应的指令空间和数据空间。程序动态分配的数据也会占用空间，而且系统为支持应用程序还要占用一定的空间。此时虚拟内存的占用情况如图 8.5 所示。

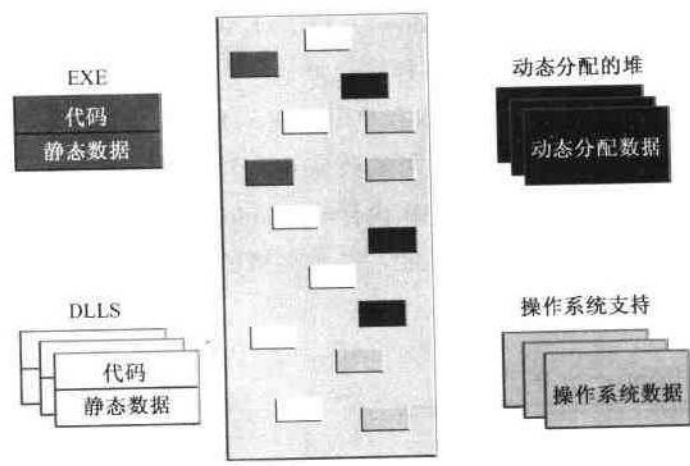


图 8.5 虚拟内存的占用情况



虚拟内存并不是一个真正存在的内存。存储在虚拟内存中的内容是分两块来存储的，一块存储在物理内存中，另一块存储在硬盘中。应用程序在运行时，如果要使用的指令或数据存储在硬盘中，操作系统首先会把它们调入到物理内存中，然后程序才能正常运行。这一点对应用程序是透明的。也就是说，应用程序并不知道操作系统在后台所做的这些工作，它只知道有 2 GB 大小的虚拟内存可以随便使用。

从图 8.6 中我们可以看出，这个虚拟内存中有一部分页面是保存到硬盘上的。如果大家使用的是 Windows 2000 的话，可以在 Windows 2000 所安装硬盘分区的根目录下找到一个名为 pagefile.sys 的文件（此文件为系统文件，默认情况下是不可见的，需要在资源管理器中设置“显示所有隐藏文件”的选项才能使其可见）。其实在这个文件中所包含的就是保存在硬盘上的虚拟内存页面。它通常比较大，会占用好几兆字节的硬盘空间。

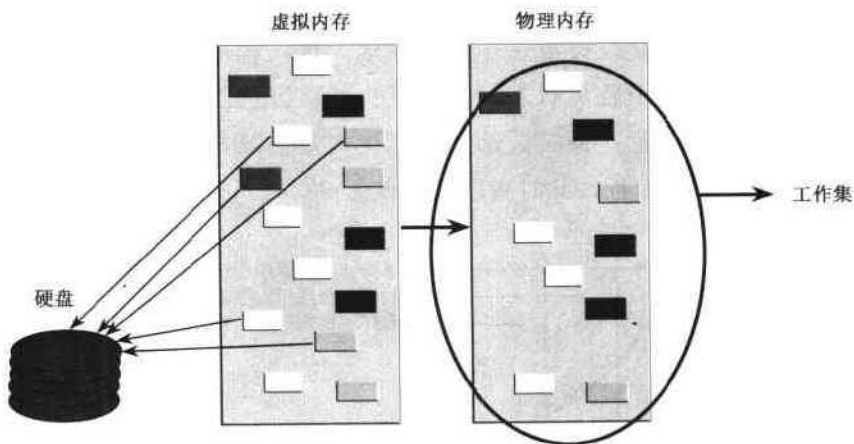


图 8.6 工作集的组成

虚拟内存中剩下的页面当然就保存在物理内存中了。实际上，物理内存中这些虚拟页面加起来的总和就是应用程序的工作集。例如，如果一个应用程序占用虚拟内存的大小为 50 MB，其中保存在



硬盘上的虚拟内存为 40 MB，则实际的工作集为 $50 - 40 = 10$ MB。

现在假设应用程序要使用一个保存在硬盘上的虚拟内存页面，则操作系统会在硬盘上找到该页面，并将其调入到物理内存中，然后程序才能运行。通常把这个过程叫做页失效（Page Fault，也可被翻译为页面错误）。

当页失效出现的时候，操作系统会在应用程序的工作集中增加新的页面，并从硬盘上删除该页面。这个过程必然会使用硬盘读写操作（Disk I/O）。所以大家在运行程序的时候经常会遇到这样的情况（特别是内存很少的时候）：程序运行过程中硬盘的指示灯总是在不停地闪烁。这就说明程序中可能已经出现页失效，操作系统正在不断地读写硬盘。

页失效对系统的性能影响非常大。图 8.7 所示列出了页失效率与内存大小之间的关系（即图 8.7 中的曲线所示）。可以看出，内存越少，出现页失效的概率就越大。其中两条直线将页失效率——“页失效数/秒”与内存大小之间的关系曲线分成了 3 个部分。最左边部分表示页失效率太高，硬盘指示灯不停地闪烁，程序无法正常运行；中间部分表示一个临界状态，此时页失效情况仍经常出现，但不影响程序的正常运行；最右边部分表示页失效情况不再出现，程序能够完全在内存中运行，此时程序的性能最高。

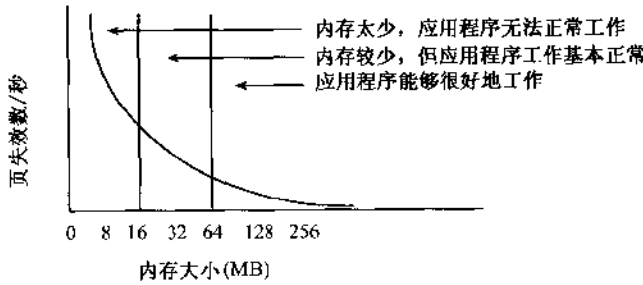
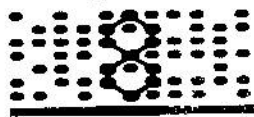


图 8.7 页失效率与内存大小之间的关系





8.3.3 查看工作集


查看工作集的方法有很多种，最常用的是使用 Windows NT/2000 系统中的任务管理器。

打开任务管理器的方法是：在 Windows NT/2000 系统中同时按下 Ctrl+Alt+Del 键打开一个对话框。在其中选择“任务管理器”按钮，即可打开 Windows 任务管理器窗口。单击其中的“进程”标签切换到“进程”面板，如图 8.8 所示。



图 8.8 Windows 任务管理器窗口的“进程”面板

在 Windows 任务管理器窗口的“进程”面板中，可以清楚地看到操作系统中当前运行的每个进程的相关数据信息。其中的“内存使用”一列显示的是当前活动的工作集；“内存增量”一列显示的是工作集的变化情况；“页面错误”（即页失效）一列显示的是页失效的总数。随着程序的运行，这些数据随时都会发生变化。




另外，使用 Windows 任务管理器窗口的“性能”面板，还可以查看操作系统的性能。

8.3.4 系统对工作集的处理过程

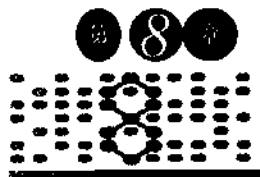
下面看一下工作集是如何变化的，即操作系统是如何操作工作集的。

操作系统会管理所有应用程序的工作集的大小。在页面切换算法中，操作系统会根据应用程序的设置来配置两个变量：**wsMax** 和 **wsMin**。其中 **wsMax** 用来表示应用程序工作集的最大值，**wsMin** 则用来表示应用程序工作集的最小值。另外，操作系统还会使用一个大小可变的变量 **wsCur**，用来表示应用程序的当前工作集的大小。通常其值位于 **wsMax** 和 **wsMin** 之间。例如，如果 **wsMax** 的值为 10 MB，**wsMin** 的值为 2 MB，则表明应用程序所允许使用的工作集的最大值为 10 MB，最小值为 2 MB，而工作集的当前值 **wsCur** 则会根据应用程序的运行情况来定，其范围通常在 2~10 MB 之间。

应用程序在运行过程中，如果出现了一个页失效，则操作系统会进行页面切换操作，将硬盘上的虚拟内存页面调入到物理内存中。在这个过程中，**wsCur** 的值会相应增大。如果应用程序所在计算机的内存很大，并且空闲内存也很多的话，则会出现一种异常情况：**wsCur** 的值可能会超过 **wsMax** 所设置的最大值。为什么会出现这种情况呢？其实这是很好理解的：由于系统中的剩余内存足够多，此时仍将 **wsCur** 的值限制在 **wsMax** 所设置的最大值之内是毫无意义的，所以还不如尽量满足应用程序的要求，提高其运行效率和性能。例如，如果操作系统中总共有 128 MB 内存，应用程序允许使用的工作集的最大值（即 **wsMax** 所设置的最大值）为 10 MB。但是，当前系统中还剩余了 100 MB 内存，因此，当应用程序请求使用 12 MB 大小的工作集时，操作系统就会灵活变通处理，分配给该应用程序 12 MB 内存空间。



另一方面，假设当前操作系统中的内存都被各种各样的应用程



序使用完了，并且你的应用程序的 `wsCur` 的值已经超过了 `wsMax` 的值，这时如果你的应用程序中出现了一个页失效，则情况就比较复杂了。应用程序在试图将硬盘上的一个虚拟内存页面调入到物理内存中时，发现物理内存已经被占满了，此时应用程序惟一能做的就是将物理内存中的某一页（这一页位于应用程序自身的工作集中）取出来放置到硬盘上，然后再把应用程序要使用的那一页放入到物理内存中。在这种情况下，应用程序的 `wsCur` 的值保持不变，即工作集的大小保持不变。显然，这种情况下出现的页失效对应用程序性能的影响比上一种情况更大，因为对硬盘的 I/O 操作更多了。更糟糕的是，如果在页失效过程中被放回到硬盘上的那一页正好又是应用程序下一次所需要的页面，则应用程序又会把工作集中的某一页放回到硬盘上，再把硬盘上的那一页调入到工作集中，而有可能应用程序下一次所需要的页面正好又是刚才被放回到硬盘上的那一页……这样，应用程序的性能就会非常低。



总结

我们可以对前面的情况进行一下总结：

- ◀ 当一个页失效出现时，新的页面会增加到当前应用程序的工作集中。如果内存很紧张，则可能会反复出现页面交换情况（即将当前工作集中的某一页放回到硬盘中，并将需要的页从硬盘中调入到工作集中），极大地影响程序的性能，并且此时工作集的大小不会变化；如果内存很富裕，则工作集的大小会增加，甚至有可能出现 `WsCur` 的值超过 `WsMax` 的值的情况。
- ◀ 页面交换过程是由 `WsMgr` 线程来实现的，它的任务就是将工作集中的页面“偷”出来放置到硬盘中，使得应用程序有更多的内存可用。`WsCur` 的值超过 `WsMax` 的值越多，则 `WsMgr` 线程从工作集中“偷”走的页面就越多。

◀ 在内存很小的计算机(如 16MB 的 Windows NT, 或 8MB 的 Windows 9x)上运行程序时, 几乎总是出现页面交换情况。

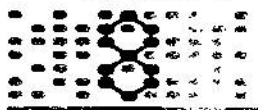
8.3.5 做操作系统的好市民

怎样才能尽量优化应用程序, 提高其性能呢? 下面列出 3 种方法。

(1) 使用 `VirtualLock()` 和 `VirtualUnlock()` 这两个 API 函数。`VirtualLock` 函数可以将申请的一段虚拟内存锁死在物理内存中, 这样, 在任何情况下操作系统也不会将这段虚拟内存页面调回到硬盘中。因此, 如果程序分配的某一段内存对程序的运行非常重要, 不希望操作系统在繁忙的情况下将它移动到硬盘上的话, 就可以使用 `VirtualLock` 函数将这段内存锁定在物理内存中。用完这段内存以后, 记住一定要调用 `VirtualUnlock` 函数。`VirtualUnlock` 函数会告诉操作系统对刚才锁死的内存进行解锁, 即操作系统在需要的时候可以将这段内存交换到硬盘中。

(2) 使用 `SetProcessWorkingSetSize(-1, -1)` 函数。当应用程序空闲时, 或者不再使用工作集时, 可以调用这个 API 函数。该函数能够释放掉应用程序的整个工作集。微软已经将这样的操作集成到了 Windows NT/2000 中: 当用户与有 UI (用户界面) 的应用程序进行交互时, 它就会生效。例如, 当用户极小化一个窗口时, 操作系统就会马上调用 `SetProcessWorking SetSize(-1, -1)` 函数, 释放掉该窗口所属应用程序的工作集, 从而使得整个系统有更多的内存可用。下节将对这种情况进行一些演示。

(3) 在编译应用程序时使用 `link/WS:AGGRESSIVE` 选项。这样,



操作系统就会知道，它可以任意地对该应用程序的工作集进行处理。操作系统在需要的时候会很快地将该应用程序所使用的内存放回到硬盘中，而应用程序在需要的时候会重新将这些内存调入到物理内存中。在应用程序空闲时，操作系统还会把该应用程序的工作集的大小调整为最小值 `wsMin`。Windows NT/2000 本身已经对 Explorer, Spooler, services 等服务进程进行了这样的处理，从而使得操作系统能够为别的应用程序提供更多可用的物理内存。

如果能够充分利用上述 3 点，对应用程序性能的提高是非常有效的。

8.3.6 演示

下面利用 Windows 2000 中的性能监视器对上述第 (2) 种方法进行一个演示。这里以 WINWORD 为例进行说明。

首先要打开性能监视器。性能监视器的执行文件位于 Windows 2000 系统目录的 `system32` 目录中，文件名为 `perfmon.exe`。执行该文件，就会打开如图 8.9 所示的性能监视器窗口。

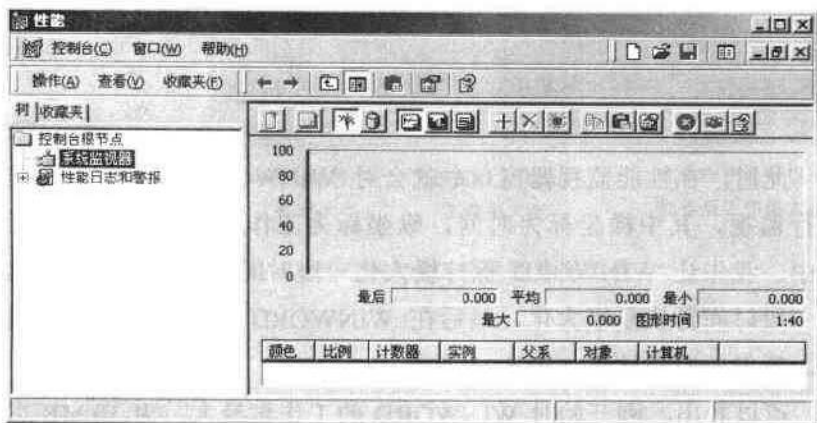



图 8.9 性能监视器窗口

假设现在已经运行了 Microsoft Word 字处理软件，并使其窗口处于极大化状态。然后在性能监视器窗口的工具条上单击  按钮，了

是会出现如图 8.10 所示的“添加计数器”对话框。在“性能对象”下拉列表中选择“Process”项，选中“从列表选择计数器”单选按钮，然后在其下面的列表中选“Working Set”项，目的是查看应用程序的记录集。接下来选中“从列表中选择实例”单选按钮。由于刚才已经运行了 Word 应用程序，因此，此时在列表中会出现“WINWORD”选项。选中它，然后单击“添加”按钮将计数器添加到性能监视器窗口中。最后单击“关闭”按钮关闭“添加计数器”对话框。

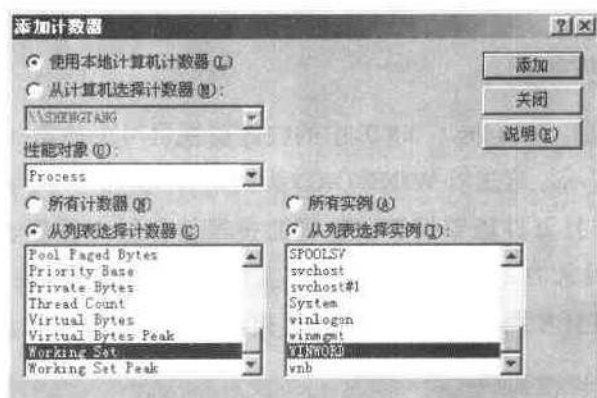


图 8.10 “添加计数器”对话框

此时，在性能监视器窗口中就会对 WINWORD 所使用的工作集进行监视，其中横坐标为时间，纵坐标为工作集的大小。在演示过程中，我先让 WINWORD 保持极大化一段时间，然后将其极小化，过一段时间再恢复极大化，最后在 WINWORD 中进行了一些相关操作。所得的监视结果如图 8.11 所示。

可以看出，刚开始时 WINWORD 的工作集最大，为 10 MB 以上（纵坐标比例尺 10:1）；当将它极小化以后，其工作集下降到最小，为 1 MB 多。这是因为操作系统对 WINWORD 做了一个 SetProcessWorkingSetSize(-1, -1) 处理，将其工作集都放回到硬盘中了；当恢复极大化以后，工作集又开始上升，增加到 4~7 MB；



当对 WINWORD 进行一些操作时，由于所需内存增加，因此工作集也相应增加，达到 8 MB 之多。

大家如果感兴趣的话，还可以对 IE, PowerPoint 等应用程序进行类似的性能监视，其结果是一样的。在我的测试中，IE 的记录集从 23 MB 降到了 6 MB，而 PowerPoint 的记录集从 7 MB 降到了 2 MB。当然，在测试中，每个人的测试结果都会有些差距，但总体趋势是一样的。

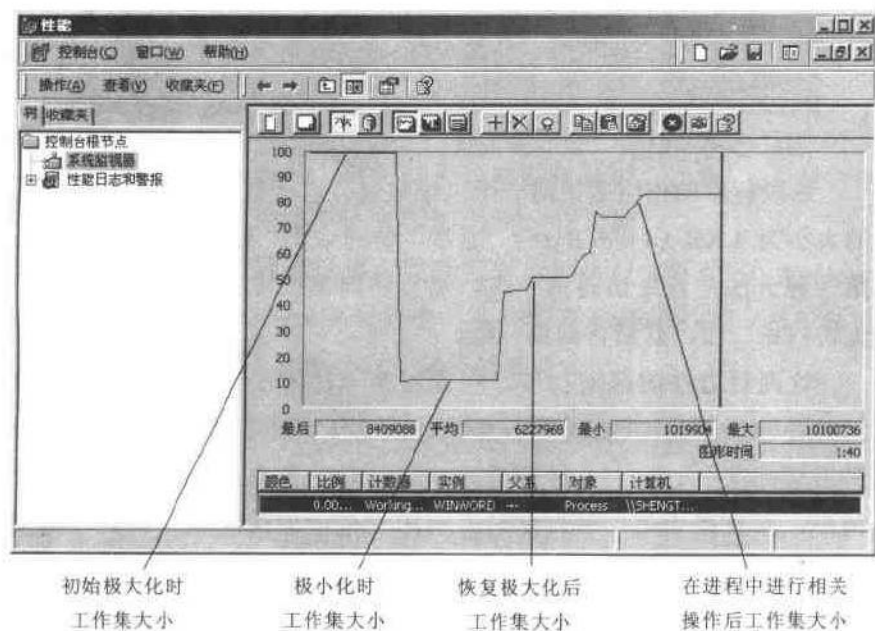


图 8.11 对 WINWORD 的性能监视结果

8.3.7 代码分析

为了进一步加深印象，我再分析两段简单的代码。第一段代码如下所示：

```
#define PAGE_COUNT 16
#define PAGE_SIZE 4096
DWORD rgdwArray[PAGE_COUNT][PAGE_SIZE];
// code to fill the array skips...
...
// access the array
for (int y = 0; y < PAGE_SIZE; y++) {
    for (int x = 0; x < PAGE_COUNT; x++) {
        DWORD dw = rgdwArray[x][y];
    }
}
```

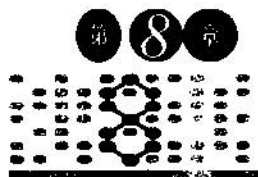
这段程序的作用是访问一个二维数组。这个数组有 16 页，每页的大小为 4 KB (4 096 Byte)。通常，访问这个二维数组有两种方法：第一种方法是首先访问第一维，然后访问第二维；第二种方法是首先访问第二维，然后访问第一维。

这两种方法的区别在于：在采用第一种方法时，程序是按照数组的存储顺序来访问的。系统会先访问第 1 页，然后访问第 2 页……最后访问第 16 页。这样，程序在运行时，基本上不会出现页失效情况；而在采用第二种方法时，程序则是前后跳动着访问的，需要在各个页中来回切换。这样有可能会产生 16×4 096 个页失效情况出现。

显然上述代码是采用第二种方法来访问数组的。为了计算系统访问数组所花费的时间，可以在上述代码的 for 循环的前后分别加上一个 GetTickCount 函数，这个 API 函数的作用就是获取时钟计数。

经过计算，采用第二种方法访问数组需花费 10 个 TickCounts 的时间。

下面我们对上述代码进行修改，如下所示。



```
#define PAGE_COUNT 16
#define PAGE_SIZE 4096
DWORD rgdwArray[PAGE_COUNT][PAGE_SIZE];
// code to fill the array skips...
--
// access the array
for (int y = 0; y < PAGE_COUNT; y++) {
    for (int x = 0; x < PAGE_SIZE; x++) {
        DWORD dw = rgdwArray[x][y];
    }
}
```

这就变成了使用第一种方法来访问数组。同样对其访问时间进行测试，结果为 0 个 TickCount，表明访问时间非常短。由此可见，页失效对程序性能的影响是非常大的。

接着来看第二个例子。这个例子是在某次中学参加数学竞赛时遇到的，代码如下所示：

```
#define MAX_LENGTH 1024*256*64
int rgLookup[MAX_LENGTH];
// code to fill the array skips...
// 1. access the array
for (i = 0; i < MAX_LENGTH; i++)
    temp = rgLookup[i];
// 2. access the array
for (i = 0; i < MAX_LENGTH; i++)
    temp = i+i;
```

当时比较流行的做法是：将乘法表的结果（或者其他需要计算的结果）直接保存在一个数组中，使用的时候直接去数组中查询，



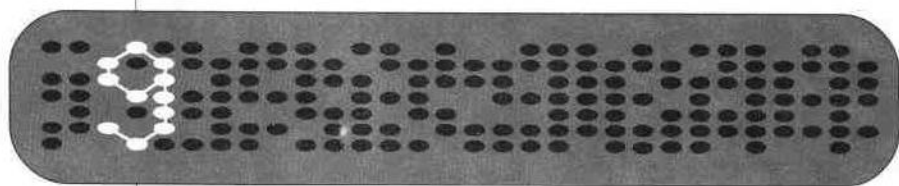
而不用再计算，从而节省 CPU 计算的时间，提高程序的运算速度和性能。例如，要想知道 8×7 等于多少，直接在数组中去查，而不用通过计算就可以得到 56 的结果。那么，究竟这种方法能否提高程序的性能呢？

在上面的代码中，我定义了一个很大的数组，然后通过两种方法来访问该数组：一是直接在数组中查询，二是现场计算。

通过 `GetTickCount` 函数的计算，可以得到每种方法访问数组所使用的时间，其结果是令人吃惊的：第一种方法的 `TickCount` 值为 581，而第二种方法的 `TickCount` 值为 300！

造成这种结果的原因是：由于数组非常大，因此在访问它时会产生页失效情况，造成系统进行硬盘 I/O 操作。尽管节省了 CPU 计算的时间，但是页失效所造成的时间消耗要远远大于 CPU 计算所花费的时间。





第9章

软件测试基础 Basic of Software Testing

背景

软件测试是一门非常崭新的学科，目前研究的内容还不很深入，所涉及的只是测试数字、测试函数等一些非常简单的问题，可以说还处于婴儿阶段。由于软件测试学科还不成熟，它到底需要一个什么样的专业基础，尚无定论，而且目前还没有一种很好的标准来衡量一名测试人员的优劣。本章，陈宏刚博士根据亲身体会，以微软公司为例讲述软件测试的方法及实施过程。陈宏刚博士指出，软件测试学的发展还有赖于大家共同努力，促进其研究的不断深入。相信本章能对软件测试工作起一定的指导作用。

同学们在认真聆听陈宏刚博士的讲座





本章内容概览

- 📁 概述
- 📁 关于 Bug
- 📁 软件测试方法和辅助工具
- 📁 相关测试文档
- 📁 如何与项目经理及开发人员沟通
- 📁 结束语

9.1 概述


由于本章所讲的软件测试是以微软公司为例，因此首先简单介绍一下微软的开发团队，然后介绍我对软件测试的一些理解。

9.1.1 微软的开发团队

当一个软件产品的研发任务确定以后，就需要组织一支有效的软件开发团队。在微软，软件开发团队由项目经理、软件产品团队和测试人员组成。

由于每一个大的产品都有几大方面的功能。因此，可以将团队中的软件研发人员分成几个软件产品团队（Software Product Teams），每个软件产品团队分别负责产品的一个功能。软件产品团队由项目经理来协调。每个软件产品开发团队一般分为三个部分：

- ◀ 项目管理团队（Program Management Team）
- ◀ 软件开发团队（Software Development Team）
- ◀ 软件测试团队（Software Testing Team）



通常，前两个团队的人数之和小于或等于软件测试团队的人数。每个团队下面再分成很多小团队，每个小团队又可以细分成很多小单元，非常灵活机动。

在一个软件产品的研发过程中，有三类重要的角色：项目经理（Program Manager，简称 PM）、开发人员（Developer，简称 Dev）和测试人员（Tester）。他们各司其职，互相支持，互相制约，以保证一个好产品的问世。


项目经理

项目经理是市场、用户与软件开发人员之间沟通的桥梁，其职责是对外做市场宣传和信息反馈，对内为软件产品的每一个特性进行清楚的定义。当开发人员遇到困难时，就要跟项目经理沟通，一同来解决问题。如果有必要的话，就需要对产品计划进行适当的调整。

项目经理的工作从开发周期上可以分为：前期、中期和后期工作。

前期

项目经理的主要任务是与一些产品部门进行交流，对外部客户做调查研究、需求分析，在一个潜在的市场里面发现做什么项目能够带来经济效益，从而决定应该去开发一个什么样的软件，或者给已有的软件增加什么样的新功能。在做出以上决策的前提下，分析其他相同软件的特性，定义新软件功能集、写功能说明性文档等。此外，负责从产品定义到中层界面，甚至到最终的用户界面等多方面的任务。





中期

项目经理的主要职责是进度控制。他要保证产品能够及时发布，为此需要在开发团队内进行合理的资源调配，并及时向上级汇报工作进度，解决一些其职权外的相关问题。项目经理还要组织员工适当地放松，调动员工的积极性，以保持较高的开发效率。在开发过程中遇到问题时，项目经理要在各部门之间进行协调，并做出折中的决定。

后期

当产品的一个版本还没有做完的时候，项目经理就要开始为下一个版本软件需要什么新的功能特性做准备，以保证软件开发的连续性和版本不断升级性。

开发人员

开发人员的职责是设计好的算法并编写程序以实现软件的预定功能。他必须严格按照项目经理的总体设计来编写程序。同时，还要保证能和其他团队保持一致。微软的软件开发人员分为两种：软件建筑师（Software Architector）和软件开发工程师（Software Development Engineer，简称 SDE）。

软件建筑师 负责软件的整体设计，以及产品的结构定义，是开发人员中不可缺少的一部分。在一个产品的开发团队内一般配备一名软件建筑师。随着开发过程的深入，软件建筑师的工作职能可能会发生变化，在后期可能会变为软件开发工程师。由于软件建筑师对整个系统比较了解，对出现的各种问题有较强的解决能力，所以在开发过程中，经常是由软件建筑师带领软件开发工程师去解决

各种问题。

软件开发工程师 职能单一，就是负责代码的设计和程序的实现。

测试人员 职能单一

微软的软件测试人员也分为两类：测试工具软件开发工程师（Software Development Engineer in Test，简称 SDE/T）和软件测试工程师（Software Test Engineer，简称 STE）。

测试工具软件开发工程师 负责写测试工具代码，并利用测试工具对软件进行测试；或者开发测试工具为软件测试工程师服务。产品开发后的性能测试（Performance Test）、提交测试（Check-in Test）等过程，都有可能要用到 SDE/T 开发的测试工具。由于 SDE/T 和 SDE 的工作都是写代码，具有相通的地方，所以两者之间互相转换的情况比较多。但需要注意的是，两者写出来的代码用途是不一样的，SDE 写的是产品的代码，而 SDE/T 写的代码只用于测试产品。

软件测试工程师 负责理解产品的功能要求，然后对其进行测试，检查软件有没有错误（Bug），决定软件是否具有稳定性（Robustness），并写出相应的测试规范和测试案例。

除此之外，在一个软件产品的研发和销售过程中，还会需要负责给产品打补丁（Service Pack）的快速修正工程师（Quick Fix Engineer），通常由 SDE 来担任，通过电话方式向用户提供售后技术支持的支持工程师（Support Engineer），销售和市场营销（Sales and Marketing）人员，研究员和研究工程师（Researchers & Research SDE）。

在进行产品开发的时候，主要是由前面三类人员（项目经理、



开发人员及测试人员)组成产品开发团队来进行的。

在微软内部,软件测试人员与软件开发人员的比率一般为 1.5~2.5 左右,这可能远远超出了大家对测试人员的理解,但微软软件开发的实践过程已经证明了这种人员结构的合理性。图 9.1 中显示了上述两个产品的微软软件开发人员的一般配置图。

下面以微软 Exchange 2000 和 Windows 2000 为例介绍一下微软产品团队的人员结构(这里只分析三类主要的人员,即项目经理、开发人员及测试人员),如表 9.1 所示。

表 9.1 Exchange 2000 和 Windows 2000 中的人员结构

	Exchange 2000	Windows 2000
项目经理	25 人	约 250 人
开发人员	140 人	约 1 700 人
测试人员	350 人	约 3 200 人
测试人员/开发人员	2.5	1.9

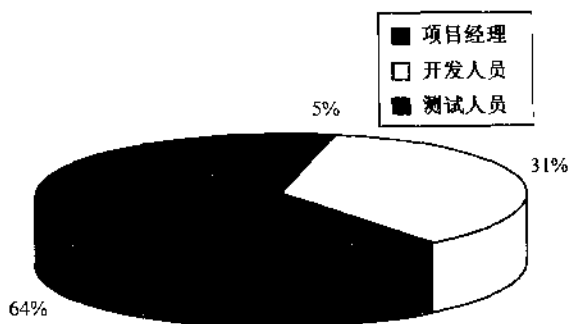



图 9.1 微软软件开发人员一般配置图

9.1.2 对软件测试的理解


由上述可知测试人员的重要性。国内的一位博士到微软总部参观之后,觉得非常奇怪:“Tester 是做什么的?为什么微软在做产品



时，测试人员占了一半以上？”其实，软件测试在产品开发中占据相当重要的一部分，这是一种需要，是微软从二十几年的实践中明白的道理，也是微软从不断的失败中总结的经验。正是由于清晰地认识到了软件测试的重要性，微软的产品质量才有了明显地提高，日臻完善，而且产品的发布速度越来越快。

大家可以感觉到，微软以前的产品有时会发生崩溃、死机等现象，而今天的产品则比五年前的产品更大、更完善，并且要稳定得多。为什么呢？这是因为我们公司的测试工作越来越好，测试人员越来越多，而且越来越有经验。关于这一点，微软公司曾经算过一笔账：最初，微软公司与大家一样，认为测试不重要，重要的是开发人员。通常，一个团队中有几百个开发人员，但只有几个测试人员，并且开发人员的工资比测试人员高很多很多。经过多年的实践后公司发现，为那些出现问题的产品再去修一个补丁程序所花的钱，比多雇用几个测试人员的费用要多得多。所以，公司就开始不断地多雇用测试人员。通过二十多年的实践，公司终于发现，根据产品的需要，测试人员应该多一些！其实，这没有任何理论根据，只是在实践中获得的经验。而且现在测试人员的工资也越来越高。测试人员水平越高，找到 Bug 的时间就越早，软件就越容易更正，产品发行之后越稳定，公司赚的钱也越多。这是微软慢慢悟出来的道理。

测试人员的任务很清楚，就是站在使用者的角度上，通过不断地使用和攻击刚开发出来的软件产品，尽量多地找出产品中存在的问题，也就是我们所称的 Bug。



可以说，找 Bug 是一个非常重要的工作，因为任何一个产品开发出来以后，都会存在许多大大小小的 Bug，轻则影响用户的正常

使用，重则导致系统崩溃。测试人员就要负责找出 Bug，并告知软件开发人员。开发人员修改、调试好程序，再交测试人员测试，去发现新的 Bug。特别是对一些大系统来说，越早发现 Bug，就越容易修复；越晚发现，就越难修复，到最后，可能只好忍痛将某个特性去掉，否则，就只能将该 Bug 留在产品里面。经过这样一个反复的过程以后，一个软件才能趋于完善和稳定，最后交付用户使用。正是因为找 Bug 如此重要，所以越大型的软件的开发，软件测试人员占整个软件产品团队总人数的比重越大，甚至要占一半以上。

但是，软件测试到底是一门什么样的科学？没有人知道，大家只是在摸索。它需要什么样的背景，也没有人知道。所以，微软的测试人员中，只有很少一部分是计算机专业毕业的，其余的有学俄语的、英语的、数学的、工程的、生物的……种类特别多，但有一点：测试人员一定要有一种感觉。

由于现在的软件测试仍然处于发展阶段，往往测试人员还是依据本能、靠感觉、靠天赋来做软件测试。如果像用户那样通过用来发现 Bug 不是真正的测试，这只是一种最基本的测试，只能发现一般用户的问题。一般来说，软件在正常使用时都不会有问题，因为在开发软件时就保证了在正常情况下的使用。但是，在实际使用中难免会出现不正常的情况。例如，用户输入了一个错误的字母或其他异常情况，这就要求软件还应该具备错误承受力，至少应该为用户提示一条错误信息。微软现在的产品都做到了这点，如果出现了错误，它会将错误信息告诉用户。但是，我用了许多国内目前的一些软件，常常得不到错误提示信息。有时因为错误操作造成软件崩溃了，同样没有向我提示错误信息；有时我提供错误的输入，这些软件也不拒绝，还会返回许多乱七八糟的结果。这些现象都是不应该的。

而且，微软产品的质量控制及产品发布日期的预测与测试领域也有关。软件测试是一门交叉学科，非常难分析它。尤其难测试的就是软件兼容性及大型软件内部的一些东西，现在还没有人能够从



理论上清楚地说明这些问题。

我习惯用军队中的进攻和防守的关系来形容软件测试和软件开发的关系。通常，士兵在防守时是可以遵循一定的规则的，如怎样布防护网，如何挖战壕，在何处布碉堡，在何处埋地雷等。但是，在进攻时则常常无章可循，往往讲究出奇制胜。只要能攻进去，不管使用什么方法都可以。

下面讲一下在测试时应考虑的几个问题。

(1) 测试最重要的一件事就是要考虑到所有的出错可能性。同时，还要做一些不是按常规做的、非常奇怪的事。

说起来可能不太好听，测试的过程就像黑客（Hacker）的攻击过程那样。可以这么说，像黑客这样的人是最好的软件安全测试员。他们专门找软件的漏洞，从而破坏这个软件，这样就可以修复这些漏洞保证软件的性能。如果找不到这种漏洞，那就说明该软件质量已经很好了。

(2) 除了漏洞之外，测试还应该考虑性能（Performance）问题，也就是一定要保证软件运行得很好，非常快，没有内存泄漏，不会出现那种越来越慢的情况。

我们可以在不关机的情况下，与其他软件一起持续运行一个多月，看看是否会出现越来越慢的情况（当然必须保证其他软件是没有问题的）。我们在做 IE 的时候，就是让它 72 小时连续不停地打开不同的网页，处理几万个不同的网页，而且速度不能减慢。有许多软件，当只有一两个人用的时候，可能感觉不到什么问题，而当几百个用户一起用的时候，有的网站就出现各种各样的异常，这就是测试工作还比较欠缺的原故。



(3) 另外，测试还要考虑软件的兼容性（Compatibility）。

一般来说，一个软件是由许多小软件构成的，如果其中一个小软件与它的前一版本不兼容，那么这个软件就会出现错误。这种错误需要通过测试来发现和解决。

许多人认为写代码的人一定能找出错误来。其实开发人员在写代码的时候，如果有错误，他可以意识到了，可是写出来的错误，他不一定能想得到。我自己也编过程序，在编程序的时候很自信，觉得不会有错，可事实上，即使是我编的小程序也有错误，但要自己找出来，却要费很大劲。因为我一直认为自己不应该出错，但常常错误就出现在我认为最有把握的地方。我是学数学的，是一个很细心的人，可是一样还是会出错，但要找出自己的错误却要花费很长的时间。后来我写的代码让我的师弟帮我看，结果他很快就找到许多问题，可是我自己花一个月时间可能都找不到。所以，开发人员和测试人员完全不一样，开发人员确实可以找到一些 Bug，但是有更多的 Bug 是他意识不到的。

在一般的开发团队中，并不需要测试人员定位 Bug 的具体位置，所以，对测试人员的要求并不高。只要你愿意学，测试工作是非常容易做的。但是，我当年所在的 IE 团队（IE 4.0）就不同，因为当时正在与另一个公司的产品竞争，所以微软就要求尽量找到一流的开发人员和一流的测试人员，尽快开发出新产品，打败对手。所以，当时对我们测试人员的要求非常严格，不仅要找出 Bug，而且要定位引起此 Bug 的代码行。然后将这些信息交给开发人员，后者就可以很快更正，省去了他们找错误出处的时间。因此，当时 IE 的开发速度非常快，一年之内就发布了一个新版本，而且几乎没有任何大 Bug，大大超越了竞争对手。

那么，什么是 Bug 呢？





9.2 关于 Bug

Bug

Bug 的定义很广泛，在软件使用过程中所出现的任何一个可疑问题，或者导致软件不能符合设计要求或满足消费者需要的问题都是 Bug，即使这个 Bug 在实践中是可行的。

有时候，Bug 并不是程序错误。例如，软件没有按照一般用户的使用习惯来运行，此时也可以把这个问题看成是该软件的一个 Bug。通常，对 Bug 的跟踪过程如图 9.2 所示。

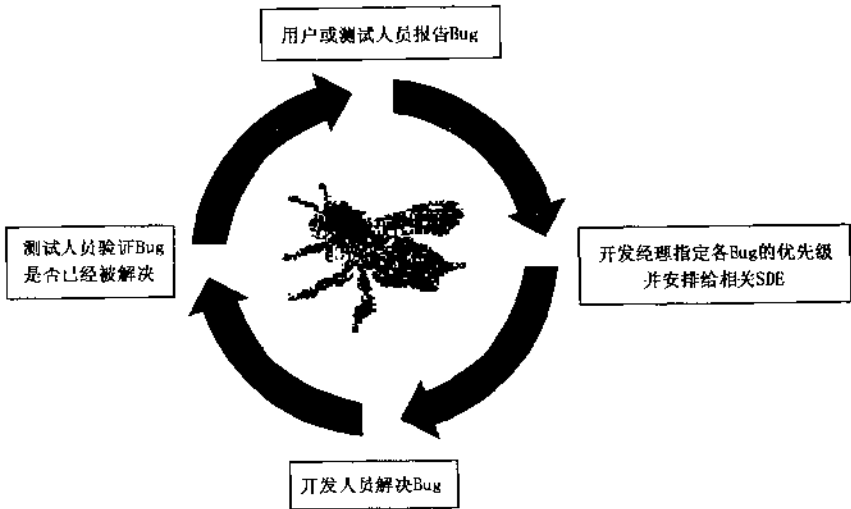


图 9.2 对 Bug 的跟踪过程



首先，测试人员根据测试结果来报告他发现的所有 Bug。通常，这项工作还需要用户的参与。微软在正式发布一个软件之前，经常要依次发布 Alpha 测试版、Beta 测试版让用户试用，以便用户能够反馈出相关 Bug 的信息。但是，现在随着微软测试队伍的扩大及测试水平的提高，越来越多的 Bug 都是我们内部的测试人员自己发现的，很少出现外部用户所反馈的 Bug 没有被测试人员发现的情况。

然后，开发经理根据这些 Bug 的危害性对它们进行排序，确定 Bug 的优先级，并安排给相关的开发工程师。

接着，开发人员根据 Bug 的轻重缓急依次修复各个 Bug。

最后，测试人员再对开发人员已经修复的 Bug 进行验证，确认 Bug 是否已经被彻底更正。

微软开发一个产品经常会遇到几十万条 Bug。随着测试人员越来越多，测试工作也越来越完善。但是，如何快速有效地追踪并修复 Bug，仍然是摆在软件测试人员面前的一个主要困难。测试产品和追踪 Bug 时最重要的是问题的定义，要清楚需要解决什么样问题，确定 Bug 的主次关系，挑选出最主要的问题，并先解决它们。例如，有些 Bug 可能会导致死机或程序崩溃，这时就要先修复它们，而另一些 Bug 可能对软件的运行影响不大，或者出现的机会很少，就可以考虑以后再修复。

可以说，没有任何一个产品没有 Bug，也永远不可能找出并修复所有的 Bug。在修复了旧的 Bug 的同时，往往又会产生新的 Bug。

根据微软的经验，每修复三到四个 Bug 一般就会产生一个新的 Bug。

这个过程就类似于数学中的“极限”的定义，尽管我们知道某个

极限值为 0，但是它永远也不可能达到 0。这也就是产品的 Bug 永远也修复不完的原因。因此，我们在修复 Bug 时要注意，一定要记住用户最需要的是什麼，然后优先尽力修复那些影响用户使用的 Bug；而对于大部分对用户影响很少、甚至根本不影响的 Bug，则可以推迟修复，甚至不修复。

在微软公司，Bug 经过开发人员解决后，再回到软件测试人员手中时，会被分为以下几种类型。

Fixed: 表示 Bug 已经被修复或更正了。

Duplicated: 表示测试人员所找到的某个 Bug 已经被别人找出来了。这种情况是很难避免的。一是因为同时有许多测试人员在进行测试，这样就很有可能多个测试人员先后发现同一个 Bug；二是因为同一个 Bug 在不同的进入情况下所表现的现象也不一样，尽管表面上看起来是不同的 Bug，但实际上可能是相同的。因此在报告一个 Bug 之前一般都要搜索该 Bug 是否已经存在。

Postponed: 表明这个 Bug 不是很重要，在当前阶段不用进行更正了；或者更正这个 Bug 风险太大，Bug 本身又不会造成大的影响。

By design: 测试人员认为是 Bug，不符合逻辑，也不符合用户的需求，但开发人员则认为是按照项目经理的设计做的。

Not repro: 以前出现的某个 Bug 自动消失了，可能是在处理其他 Bug 的时候把这个 Bug 一并修复掉了。

Won't fix: 这个 Bug 是一个错误，还没有重要到非要更正不可的地步，完全可以忽略不计。



在查找并修复 Bug 的过程中,测试人员和开发人员经常会发生争执。例如,当开发人员宣布某一个 Bug 已经被 Fixed 时,测试人员往往还不放心,又重新对该 Bug 进行检查;当开发人员宣布某一个 Bug 是 Duplicated 时,细心的测试人员也要验证该 Bug 是否和另外一个 Bug 相重复,如果另外一个 Bug 已经被修复了,但这个 Bug 还存在,就会让开发人员重新修复该 Bug;对于是否需要 Postponed 一个 Bug,测试人员和开发人员也常常争论不休,测试人员认为这个 Bug 需要修复,而开发人员则认为修复这个 Bug 不值得,这时候就需要项目经理来决定,因为测试人员要从用户的角度来看待 Bug,开发人员则要考虑到开发期限和付出的代价,而项目经理要同时考虑到这两种情况。

只有项目经理经过权衡之后才能确定是否要推迟修复该 Bug;在 By design 情况下,通常是争论最多的时候。这时候也需要三方都坐下来谈,其结果一般有三种:坚持原来的设计,修改原来的设计并增加特性,或者取消该设计。对 Not repro 和 Won't fix 情况也是这样,需要测试人员、开发人员和项目经理进行协商,直到三方达成共识。

9.3 软件测试方法和辅助工具

有了 Bug 类型的定义以后,如何去找出这些 Bug 呢?这就需要采用好的测试方法。以下介绍几种常用的软件测试方法。

有多种方式对软件测试方法进行分类。例如,从代码和用户使用的角度可以将软件测试方法分为如下几种。

(1) 覆盖性测试 (Coverage Testing)

覆盖性测试是从代码的特性角度(即内部)出发的测试方法,包括以下方式。

- ◀ 单元测试 (Unit Test): 按照代码的单元组成逐个进行测试。
- ◀ 功能测试 (Function Test) 或特性测试 (Feature Test): 按照



软件的功能或特性逐个进行测试。例如，在 Exchange 中，发送邮件和接收邮件就是两个不同的功能或特性，在测试时就分别对它们进行检查，看是否工作正常。

- ◀ **提交测试 (Check-in Test)**: 在开发人员对代码做了任何修改，或者修复了某个 Bug 时，需要重新 Check-in 代码（即将修改后的代码放入到整个大的系统中）。这时，开发人员往往也要进行测试，看代码是否工作正常。为了保险起见，开发人员往往要找测试人员帮着一起进行测试（我们把这种情况称做 **Buddy Test**）。测试人员和开发人员之间搞好关系是非常重要的，稍后我会专门讲述这一点。
- ◀ **基本验证测试 (Build Verification Test, 简称 BVT)**: 对完成的代码进行编译和连接，产生一个构造，以检查程序的主要功能是否会像预期一样进行工作。这是最简单而又最省时的一种测试方法。每产生一个新的构造时都要进行测试。如果连 BVT 都通不过，表明问题很严重，开发人员需要尽快修复出现的问题，测试人员也就不必浪费时间做其他测试了。
- ◀ **回归测试 (Regression Test)**: 过一段时间以后，再回过头来对以前修复过的 Bug 重新进行测试，看该 Bug 是否会重新出现。

(2) 使用测试 (Usage Testing)

使用测试是从用户的角度（即外部）出发的测试方法。它也包括许多类型。

- ◀ **配置测试 (Configuration Test)**: 从用户的使用出发进行多方面的测试。例如，保证软件不仅能够在 Windows 9X 下运行，也能够 Windows ME 下运行，还能够在 Windows NT/2000/XP 下运行；或者软件不仅能够在配置高的计算机上运行，也能够配置很低的计算机上正确地运行。总之，要考虑到用户的多种情况，用多种配置对软件进行测试。
- ◀ **兼容性测试 (Compatibility Test)**: 主要考虑兼容性问题，比





如同一个产品的不同版本（如 Office 2000 和 Office XP）之间的兼容问题，不同厂家的同一个产品（如 IE 和 Netscape）之间的兼容问题，不同类型软件（如 IE 和 Office）之间的兼容问题等。最难测试的往往就是软件的兼容性问题，往往要投入巨大的人力和物力。一些厂商开发出来的产品在兼容性上做得很不好，就是因为没有足够的人力和物力进行测试。

我在做 SQL Server 的 XML 测试的时候，为了解决 XML 的兼容性问题，用了 6 个测试人员和 100 台计算机进行测试。正因如此，微软产品的兼容性都非常好。而不像市场上的一些产品，安装以后就导致计算机上的许多其他软件无法使用，或者出现各种各样的问题，这样不仅伤害了其他软件，也伤害了用户。

◀ **强力测试 (Stress Test):** 在各种极限情况下对产品进行测试（如很多人同时使用该软件，或者反复运行该软件），以检查产品的长期稳定性。例如，我们在开发 IE 4.0 的时候，由于当时有一个非常强的竞争对手，因此我们必须保证 IE 4.0 要做得非常好。当时，为了测试 IE 4.0 的长期稳定性，我们专门设计了一套自动测试程序，它一分钟可以下载上千个页面。我们使用这个测试程序对 IE 4.0 进行了连续 72 小时的测试，也没有出现任何问题，如内存泄漏、程序崩溃等。

强力测试时间要求

根据微软的实践经验，如果一个软件产品能通过 72 小时的强力测试，则该产品超过 72 小时后出现问题的可能性微乎其微。所以，72 小时就成为微软产品强力测试时间的标志 (Benchmark)。



本项测试可以帮助找到一些大型的问题，如死机、崩溃、内存泄漏等，因为有些存在内存泄漏问题的程序，在运行一两次时可能不会出现问题，但是如果运行了成千上万次，内存泄漏得越来越多，就会导致系统崩溃。

- ◀ 性能测试 (Performance Test): 本项测试是保证程序具有良好的性能。如果别人的产品只需 5 秒钟就能得出结果，而你的产品需要 10 秒钟才能得出结果，就说明你的产品性能不好。如果在测试过程中发现性能问题，修复起来是非常艰难的，因为这常常意味着程序的算法不好，结构不好，或者设计有问题。因此在产品开发的开始阶段，就要考虑到软件的性能问题。
- ◀ 文档和帮助文件测试 (Documentation and help file Test): 因为用户通常是通过文档和帮助文件来学习使用产品的，如果文档和帮助文件存在错误，就可能会导致用户无法正常使用产品。这项工作通常在产品即将 Ship (即准备包装发布) 时进行，以避免在修复 Bug 的过程中需要反复修改文档，或者忘记修改文档，导致文档与产品的特性不相符。
- ◀ Alpha 和 Beta 测试 (Alpha and Beta Test): 在正式发布产品之前往往要先发布一些测试版，让用户能够反馈出相关信息，或者找到存在的 Bug，以便在正式版中得到解决。

还有一种分类方法将测试方法分为如下几种。

(1) 白盒测试 (White Box Testing)

又叫做玻璃盒测试 (Glass Box Testing)。在软件编码阶段，开发人员根据自己对代码的理解和接触所进行的软件测试叫做白盒测试。这一阶段测试以软件开发人员为主，有时候 SDE/T 也会辅助开发人员进行测试。





(2) 黑盒测试 (Black Box Testing)

黑盒测试的内容主要有以下几个方面。

- ◀ 接受性测试 (Acceptance Testing): 类似于 BVT 测试。
- ◀ Alpha/Beta 测试 (Alpha/Beta Testing): 在此过程中, 产品特征不断地修改。当发现 Bug 后, 在开发人员修改的同时, 项目经理也会对产品计划做出相应的调整, 产品计划不是一成不变的。
- ◀ 菜单/帮助测试 (Menu/Help Testing): 大家千万不要以为这一项测试不值得进行。其实, 在软件产品开发的最后阶段, 文档里发现的问题往往是最多的。因为在软件测试过程中, 开发人员会修复测试人员发现的 Bug, 而且可能会对软件的一些功能进行修改, 同时项目经理也会根据情况调整软件的特性, 因而在软件开发和测试的过程中, 所有的功能和特性都不是固定不变的, 都会进行调整。所以, 一般来说, 直到软件 Ship 时才编写软件的帮助文档, 这样才能保证帮助文件的内容与软件功能相符。我在做帮助文件测试的时候, 总是假装什么都不懂, 就按照帮助文件提供的步骤去做, 看看该文件是否正确。在实际测试中, 我经常能发现帮助文件的 Bug。
- ◀ 发行测试 (Release Testing): 在正式发行前, 产品要经过非常仔细地测试。除了专门的测试人员外, 还需要几千个甚至几千万个其他用户与合作者通过亲自使用来对产品进行测试, 然后将错误信息反馈给我们。到了发行测试这一步, 如果出现非改不可的 Bug, 就必须推迟软件的发行。有的时候一推就是几个月, 其间需要重新对软件产品进行全面地测试, 耗费大量的时间和人力物力。
- ◀ 回归测试 (Regression Testing): 回归测试的目的就是保证以前已经修复的 Bug 在软件 Ship 以前不会再出现。实际上, 许多 Bug 都是在回归测试时发现的。在此阶段, 我们首先要检查以前找到的 Bug 是否已经更正了。值得注意的是, 已经更



正的 Bug 也可能又回来了, 有的 Bug 经过修改之后可能又产生了新的 Bug。所以, 回归测试可保证已更正的 Bug 不再重现, 不产生新的 Bug。

◀ RTM测试(Release To Manufacture Testing): 为产品真正的 Ship 做好准备所进行的测试。事实上, 在这一测试阶段, 对每一个 Bug 都需要经过很高职务的人同意才能更正。因为这时候修改软件非常容易产生其他的错误, 所以只有那种非修复不可的 Bug 才会允许进行修改。如果在发行阶段软件还有许多严重的 Bug 的话, 恐怕就不能按时发布了。记得有一次一个微软核心产品刚刚完成, 准备 Ship 时, 我对其进行 RTM 测试时就发现了一个 Bug: 只要用该产品打印中文就会导致程序出错。这是一个很严重的 Bug, 于是开发人员马上修复了该 Bug, 重新 Ship 该产品。

◀ 功能及系统测试 (Function & System Testing): 这一点是最重要的, 它包括了非常多的内容。

- 规范验证 (Specification verification)
- 正确性 (Correctness)
- 可用性 (Usability)
- 边界条件 (Boundary condition)
- 性能 (Performance)
- 强力测试 (Stress)
- 错误恢复 (Error Recovery)
- 安全性 (Security)
- 兼容性 (Compatibility)
- 软件配置 (Configuration)
- 软件安装 (Installation)

另外一种分类方法就比较好理解了, 主要将软件测试方法分为如下几种。

(1) 手工测试 (Manual Testing)





即依靠人力来查找 Bug。

(2) 自动测试 (Automation Testing)

即编写一些测试工具，让它们自动运行来查找 Bug。

自动测试的优点是能够很快、很广泛地查找 Bug，缺点是它只能检查一些最主要的问题，如崩溃、死机，但是却无法发现一些一般的日常错误，这些错误通过人眼很容易找到，但机器却往往找不到。另外，在自动测试中编写测试工具的工作量也很大，因此在实际测试中通常是手工测试和自动测试相结合，而且手工测试往往是主要的，占了 $1/2 \sim 2/3$ ，而自动测试只占 $1/3 \sim 1/2$ 。在不同的开发队伍中，这个比例会有所不同，但总体趋势是这样的。

自动测试案例一

我在 Exchange Server 团队的时候，常常要做一种 Stress Test，需要几万个甚至几十万个用户同时把 E-mail 发送到服务器 (Server)，以保证 Server 不会出现死机或崩溃的现象。可是，需要几万人同时发送 E-mail，这在现实生活中很难人为实现。但是，利用测试工具就可以非常容易地做到。测试工具可以自动产生几万个账号，并且让它们在同一时间从不同机器上（一个机器上可以有多个不同的账号）同时发送 E-mail 信息。

自动测试案例二

再举一个浏览 Web 页面的例子，要求 50 000 个用户同时浏览一个 Web 页面，以保证网站的服务器 (Server) 不会死机。一般来说，找到 50 000 个用户同时打开一个网页是不现

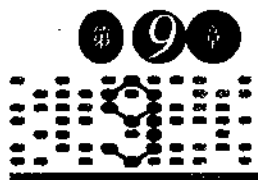
实的，就算能够找到 50 000 个测试者，成本也非常高。但是通过测试工具则很容易做到，并且工具还可以自动判断浏览结果是否正确。

有了好的测试方法，还需要有高效好用的辅助工具。做软件测试通常需要以下一些基本工具。

- ◀ 计算机
- ◀ 优秀的办公处理软件（如字处理软件和表单软件，用于编写测试计划和规范）
- ◀ 视频设备
- ◀ 秒表（计算程序运算时间，测试产品性能）
- ◀ 错误跟踪系统（微软内部使用的是 RAID，用来自动跟踪 Bug）
- ◀ 自动测试工具（产生 Automation 脚本）
- ◀ 软件分析工具
- ◀ 好的操作系统（如 Windows NT/2000，其中有很多有用的工具，如文件比较器、文件查看器、文件转换器、内存监视器等）
- ◀ 多样化平台

9.4 相关测试文档

微软的产品越来越庞大，功能越来越多，同时也越来越复杂，因而可能出现的问题就会越来越多，并且这些问题也越来越不容易被发现。另一方面，由于现在的软件测试仍然处于发展阶段，测试人员往往是依据本能、靠感觉来做测试。因此，在软件测试过程中



测试人员需要撰写一些与测试相关的文档。

（1）测试计划（Test Plan）

制订一个完整、规范的测试计划对每一个测试管理人员来说是非常重要的。当然，根据每一个测试管理人员具体任务的不同，他们的测试计划也会有所差异。

测试计划和产品开发紧密相关，由好几个部分组成。所有大型的商业软件都需要有完整的测试计划，需要具体到每一个步骤，并且每一个部分都必须符合规范要求。

（2）测试规范（Test Specification）


所谓测试规范，是指为每一个在测试计划中确定的产品领域所写的文档，用来描述该领域中的测试需求。

在编写测试规范之前，需要参考项目经理写的产品规范，以及开发人员写的开发计划。但每个开发人员习惯都不一样，有的习惯写开发计划，这样我们写测试规范和测试计划就很容易了；而有的开发人员只是让你参照项目经理的规范，这时就需要费点精力。另外，每一个领域都应该有一份详细的测试规范，所以还需要参照测试计划，因为测试规范的内容要与该计划相吻合。

（3）测试案例（Test Case）

所谓测试案例，是指描述如何测试某一个领域的文档，这些文档符合测试规范中的需求说明。

在开发测试案例之前，必须具备一份正确的项目经理规范和一份详细的测试规范。在开发测试案例时，第一个案例一定是根据规范中定义的测试想定（scenario）而开发的。在运行的过程中，根据测试反馈信息，我们可能会发现未考虑到的新问题，这就需要不断地添加测试案例。另外，当发现新的 Bug 时，也必须添加新的测试案例，这样，就可以免去回归测试。



(4) 测试报告 (Test Report)


在测试的过程中，测试管理人员会定期汇报测试进展。通常，测试管理人员以测试报告的形式向整个产品开发部门报告测试结果及所发现的 Bug。撰写测试报告的目的是为了让整个产品开发部门了解产品开发的进展情况，以使 Bug 能够迅速得到修复。

(5) Bug 报告 (Bug Report)

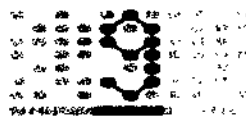
测试人员在测试的过程中，如果发现了 Bug，就应该及时向开发人员报告。通常，测试人员是以 Bug 报告的形式向开发人员报告所发现的 Bug 的。撰写 Bug 报告的目的是为了使 Bug 能够迅速得到修复，因此，测试人员的 Bug 报告撰写的好坏会直接影响到开发人员对 Bug 的修复。如果测试人员的 Bug 报告撰写得不好，开发人员就可能不能有效地从测试报告中得到关于 Bug 的正确而详细的信息，从而导致 Bug 推迟甚至没有得到修复。

在这里我只对几种测试文档作了简单的介绍。为了让大家能够更深入地了解测试文档的内容和作用，将在第 10 章用专门的篇幅讲述如何撰写测试的相关文档，并且会给出我自己亲手撰写的一些测试文档实例，供大家参考，相信大家也会非常感兴趣。

9.5 如何与项目经理及开发人员沟通



前面已经提过，在微软总部，软件产品团队由三部分组成：项目经理、开发人员和测试人员。每一部分都是一个小团队，由各自的经理 (Manager) 进行管理，这些经理是平级的，各自将结果报告给产品单元经理 (Product Unit Manager)。这三部分是独立的，测试人员绝不会归开发人员团队的经理管理，也不归项目经理所在团队的经理管理。因为这种分工的目的是为了保持一种平衡，以确保能做出一个高质量的产品。他们各司其职，但互相之间也需要经常交



流。开发人员必须保证代码的正确性，测试人员必须查出代码存在的所有问题，项目经理必须保证产品可以用，并受到用户喜欢。其实，项目经理更像是用户和产品之间的一座桥梁。项目经理包括两部分，其中一部分就是用户（Customer）项目经理，专门面向用户的，他们的工作就是做市场和产品需求分析；另外一部分负责与测试人员和开发人员交流，并在出现问题时进行协调并做出决定，以保证产品开发顺利按时完成。

假设测试人员归开发人员的经理管理，如果该经理希望能尽快交出产品，他对测试人员说此 Bug 不重要，测试人员会怎么做呢？因为你是他的手下，恐怕只能服从。所以，管理机制中测试和程序开发部门的独立性保证了测试的客观性。各部门各司其职，一个专门负责编制代码，一个专门找问题，一个负责与客户/各开发团队进行交流，从而有效地保证了产品的功能。当然，部门的独立并不阻碍他们之间的交流，项目经理、开发人员与测试人员都要有团队合作的意识。

◀ 同一个产品（Same Product）

虽然工作不一样，但目标一致，我们都明白大家都是在做一个产品，都希望做出一个高质量的产品。

◀ 一个大团队（One Big Team）


我们是一个大团队，大家要同舟共济，就像一个大家庭一样。

◀ 荣辱与共（Succeed and Fail Together）

我们应该荣辱与共，成功或失败都是一起承担的。

◀ 良好的沟通（Well Communicated）

当然，在工作过程中，开发人员和测试人员之间经常会发生摩擦，如测试人员认为这是一个 Bug，但开发人员则认为这不是什么问题。如果二者不能达成一致的话，可以找到项目经理，后者根据用户的需求或技术难度来决定是否修复。如果一个 Bug 比较严重或者影响使用的话，一般都会修复此 Bug。但是，并不是每个测试人员找到的 Bug 都是对的，测




测试人员必须认真找出真正会影响程序功能的 Bug，不要随便就喊“狼来了”。否则，以后没有人再重视你的测试结果了。

我的做法是：如果找到一个 Bug，我会先通知开发人员一声，说我找到一个很严重的 Bug，哪天有空你看一看。他们常常会立刻与我一起寻找产生这个 Bug 的原因，然后我再把错误报告（Bug Report）输入到 Bug 库中。这样，他们就觉得我很尊重他们，而且在我正式报告 Bug 时，他们大多数情况下都已经知道了产生 Bug 的原因，因此能非常快速地修复它们。于是老板对他们的工作就会很满意，他们内心里也会非常感谢我，自然我们就会逐步建立起一种很融洽的工作关系。其实，如果能解决一个 Bug，开发人员会非常开心的，因为能找到问题并解决它，说明他的水平是很高的。

可能有人 would 认为，如果一段代码的 Bug 越多，说明它的开发者水平越低。其实，在实际开发中，情况不是这样的。对测试人员来说，找到的 Bug 越多，说明他（她）的水平越高；但对开发者来说，就不能简单地依照 Bug 数目来评判其水平的高低，因为很难判断这是他（她）自己的错误，还是其他人的。例如，在实际工作中，我经常遇到这种情况：尽管对每一个程序包进行单独测试都没有任何问题，但是将它们连接在一起时，却出现了问题。此时，你很难说这是哪个开发者的问题。所以，如果开发者能够找出 Bug 并修复它，说明他（她）的水平高。尤其是在微软，开发的都是大型软件，一般都是由几百个工程师一起开发产品，根据修复 Bug 的能力就能看出开发者的水平。

（1）巴迪测试（Buddy test）

为什么叫“Buddy test”呢？“Buddy”就是关系很密切的意思。一般来说，开发人员并不知道自己即将提交（check-in）的代码里面是否存在 Bug，这时他通常会找一个关系很密切的测试人员帮他测试一下。这种帮助纯粹是义务的，因为开发人员还没有 check-in，所以找到的 Bug 是不能记录在 Bug 库里面的。我做过许多这种事情，我觉得很值得，因为人家都是好朋友嘛。开发人员当然也很开心，





因为他们还没有 check-in 就已经修复 Bug 了，这样别人就不知道原来存在 Bug。所以他们自然也对我很友好。

(2) 友好的关系 (Friendly relationship)

我个人的观点是：当我与他们的关系很友好时，很容易解决问题。当我从原来的团队调到 IE 团队以后，原来的团队成员很怀念我，对我的新老板说：“George 走了以后，我们都没有 Bug 需要修复了。”确实，当时我在那里工作时，每天我都给他们找出十来个 Bug。而其他六个测试人员找的 Bug，加起来可能还没有我找的多，特别是在产品快要发布的时候。

一般来说，产品在发布 (ship) 时往往已经很稳定了，很难再找到 Bug。在我要离开 Exchange 的一个团队的时候，该团队的老板跟我说：“每次我们的产品在 ship 时，五六十个测试人员所找到的 Bug 中，一半都是你找的。”尽管我所找到的大部分都是非常小的 Bug，但仍然给 Exchange 产品的高级管理人员留下了很深刻的印象。

另外，我认为我能找到这么多的 Bug 还有一个特点，就是与项目经理和开发人员的关系特别好。所以，一般项目经理对产品的功能做了一些改变后，都会来找我：“你赶快去测试开发人员的代码，肯定能找到 Bug。”而开发人员在修复一个 Bug 之后，也会来找我：“George，我刚刚 check-in 了……你看看有没有 Bug。”大家对我都很友好，我找到什么问题，也会马上告诉他们。

(3) 测试是独立的 (Testing is independent)

“Independent”的意思是测试人员应该有自己的观点。如果测试人员永远没有自己的观点，总是跟着别人走，那么根本没有人会关注你的建议。

记得我被提升为 IE 团队的测试组长 (Test Lead) 不久, 我的老板让我代表测试团队参加三人 (Triage) 小组, 该小组的成员包括项目经理团队、开发团队和测试团队的各一名组长 (Lead) 或经理 (Manager)。

该小组将决定在产品开发的后期, 哪些 Bug 需要修复。我刚去的时候, 项目经理组组长 (PM Lead) 和开发组组长 (Dev Lead) 根本就不重视我, 常常不顾我的反对就做出推迟或者不修复 Bug 的决定。我知道自己还没有准备好, 所以我也只是做自己应做的事。他们说什么, 我也不多反驳, 因为我知道现在还没有威信 (Credit) 让他们听我的。如果 Bug 的确不影响使用, 或者修复起来非常困难, 我也就放弃争辩。但是如果 Bug 造成的后果比较严重, 我就会找到关系比较好的相关开发人员和测试人员进行研究, 找到最简单的 Bug 重现步骤及修复办法。然后我就会找到他们, 向他们演示 Bug 是多么的容易发生, 以及修复该 Bug 对产品的其他功能产生影响的可能性很小。

这样, 由于开发工程师和测试工程师们的支持, Triage 的另两位组长经过仔细思考之后, 通常都会觉得我说的很有道理。这种情况发生多次以后, 他们就慢慢地听我的意见了, 并让我做最后的决定。

其实, 这里还有一个小小的插曲。



一个教训

在 Triage 小组开始工作两个星期以后，我的老板跟我说：“在 Triage 的决定过程中，据说你们三方中测试方面没有声音，全是开发组长和项目管理组长来做决定？这是真的吗？”我说：“确实是。但是我保证一个月以后，你听到的都是测试人员的声音！”果然，一个月后，在所有的问题上，其他两个组长都听我的。结果，每次讨论问题时，其他两个组长都会问我：“你觉得应该怎么办？”我马上说出我的看法，他们都说：“好！好！”我说：“你们再考虑考虑。”他们就说：“这件事情听你的。”我当然非常高兴。因此，我的经验是通过周密和细心的工作来赢得自己的信用。

所以，作为一名测试人员，你一定要有自己的声音。当然，最重要的一点是：你不要轻易下结论，你的声音一定要正确、一定要有价值。如果你的意见没有价值，总是说一些废话，以后你就不用说话了。

(4) 保证软件功能的定义有意义 (Make sure the feature definitions make sense)

项目经理负责定义软件的功能。这项工作是很困难的，不仅需要用户的反馈，而且还需要参照许多自己内部人员的设计，并不是面壁就能写出来的。但是值得一提的是，项目经理设计的这些功能可能并不是都很合理，因为他需要考虑技术实现的可能性。

作为一个测试人员，因为大部分时间都像是一个用户在使用软件，所以如果觉得软件的特性有什么不合适的地方，就应该去找项目经理，并告诉他你的想法。如果我觉得某个特性不合适的话，我

就去找项目经理，告诉他这个特性不合理，并解释理由。他听了之后，觉得有道理，就与开发人员一起讨论，如果大家认为我的看法是正确的，而且认为修改软件也不是很困难，开发人员就会修改软件。这样，这个功能就修改了，从而更正了这个功能（Feature Bug）（通常我们把项目经理所设计的不合理的功能称为 Feature Bug）。其实，这种 Bug 已经超越了一个测试人员的责任，因为你是在帮助项目经理修改功能上的错误。但是，我认为，作为一个测试人员，不能只是被动地去做一些测试，而是应该主动地去做一些事情。你一方面要保证软件的质量，另一方面更要保证软件的合理性。

（5）学会说不（Learn to say “no” if you strongly feel so）

作为一名测试人员，如果我强烈地认为软件的某一个 Bug 不容忽视，就必须指出并要求开发人员改正。否则，我觉得会伤害用户，即使推迟软件的发行，也要求他们修正。

案例

记得在 Windows 98 英文版本发布的当天，我找到了一个 Bug，该版本在打印中文时一塌糊涂，我就去找负责人。他们研究之后，觉得这真是一个很大的 Bug，但是他们认为中国市场并不主导我们的利润（Profit），所以觉得没有必要修正，但我坚持自己的观点，认为中国市场虽然并不主导我们的利润，但是中文用户很多，所有的中文用户一定会抗议的。后来，他们考虑了很久还是决定更正该 Bug，并因此而推迟了软件的发布时间。

商业当然是以赢利为目的，但是作为测试人员，我们应该以用户为目的，一定要坚持正确的观点。所以，我们应该表达出自己的



观点，让别人能够听到。

在实际工作中，这种坚持往往会得罪人，但最终你会赢得别人的尊重。他们会觉得你这个人很有主见，而且做事情考虑得很周全。

所以，我认为对一个团队来说，首先，项目经理、开发人员和测试人员的目标都是一致的，都是在做同一件事情，大家要互相帮助；第二，该坚持的事情一定要坚持；第三，除了做功能测试以外，还应该做异常测试，甚至在你觉得你有一个好的想法时，也应该主动提出来。微软有一个最大的优势就是允许和支持你这么做。所以，只要是正确的观点，如果你的老板不同意，你可以再向他的上级汇报，甚至汇报到比尔·盖茨那里。在微软，大家在工作中很好相处，私下里关系也很好。

(6) 项目经理定义的规范也是可以改变的 (PM's spec is changeable, too)

在产品的开发过程中，由于市场情况的变化，技术问题的出现，或者用户需求的变更，有时项目经理所定义的规范不得不进行修改。不过项目经理一般要经过协调才能做出修改决定，并将修改的内容通知所有的产品开发团队。

(7) 坚持正确的看法 (Insist what is right)

对自己认为正确的，应该坚持，不能人云亦云。这样做在开始时可能会得罪人，但最终会赢得尊重。

(8) 职业化 (Professionalism)

在工作过程中，出现意见不一致时，不要带有感情色彩，义气用事，应就事论事；还有，应该分清工作和生活的关系。

(9) 向项目经理和开发人员反馈 (Give PM/Dev feedbacks)

作为测试人员，我们需要从用户的角度来衡量一些设置，主动地提出建议。最重要的事是，我们必须向项目经理及开发人员提供

许多测试反馈结果。

9.6 结束语

作为测试人员，我个人的经验是：

第一，我是学数学的，逻辑思维比较强，因为学数学时，为了证明一个问题，必须考虑到所有情况。

第二，我自己喜欢琢磨，不喜欢循规蹈矩，我觉得这是做测试比较有效的手段之一。

在微软有一种测试方法，称之为 Ad_hoc testing，即对软件进行随机地测试的测试者。

案例

我刚进 IE 团队时，因为我已经有很多测试经验，第一天就给他们找到许多 Bug。以后，每天我都能给他们找到很多 Bug。但是，过了两个星期，我发现我的经理从来没有找过我谈话。于是，我就去找他，我说：“我来了两个星期，你都没有找我谈话，为什么？”他说：“我还需要找你吗？”我问：

“为什么？”他说：“每天我都发现你能找到很多 Bug，我不再用再给你安排什么工作，你就按照这种方式做。”但是，对那些工作经验不够的测试人员，经理就会经常告诉他应该怎么做。经理认为我已经知道怎么做了，而且我比大家找得快，找得多，所以他就不用给我安排工作。因此，在那个时候，我就相当于一个 Ad_hoc tester。这就是微软的一个优势，它能够随机应变，根据不同人的能力来做不同的调整。



最后，我以一段有趣而又真实的故事结束我的演讲。

我在微软总部做 Test Lead 和 Manager 时，经常需要招聘测试人员。我招聘人时通常会遵循两个原则：一是要看他的逻辑性，这是非常重要的一点；二是要看他的思维是否怪异，是否能经常产生一些怪点子。第一点是可以依靠后天培养的，但是第二点恐怕是后天培养不出来的，多少有些天生的因素在里面。

这两个原则是我在多年的实践中得出来的。由于软件测试学科还不成熟，目前还没有一种很好的标准来衡量一个测试人员的优劣。而且它到底需要一个什么样的专业基础，尚无定论。从目前来看，并不是说学计算机专业的就最适合做软件测试人员。


一个有趣而真实的故事

说起来也许大家不会相信，我曾经亲自招聘过一位家庭主妇来做微软的测试人员！

这名家庭主妇已经四十多岁了，是一位海军军官的妻子，三个孩子的母亲。她只读到高中毕业，连大专也没有上（在美国没有上过大专的人是很少的）。她使用计算机的水平也非常初级，而且还是跟着自己的女儿学的。后来她在家闲得无聊，就决定出来找一个工作，而且居然跑到微软来应聘了。

当时，我在面试她的时候就已经发现她的计算机水平很有限，只能达到一个一般用户的标准。但是，我发现她的思维很怪异，怪点子很多，能够很快地发现一些问题。于是，我就随便让她试用一下 IE，结果她当场就给我找出了好几个 Bug。她完全是凭着一种感觉来找 Bug 的。


后来，我对老板说，我想雇用她。老板一听，睁大了双眼：“Are you crazy? You want to hire a housewife!（你疯了吗？你居然想雇用一个家庭主妇！）”



于是我跟他说了一下我的想法。他还是觉得让一个大学都没有上过的家庭妇女做测试人员是不可思议的。但是，最后他还是说：“You are the hiring manager. You make the decision! (你是招聘经理，还是你自己来决定吧!)”

我最终还是决定雇用她。在开始阶段，她的确存在许多问题。我只是认识到她的感知力很强，却忘了她的其他素质如何。由于她一直做家庭妇女，没有在职业环境中待过，因此显得很粗鲁，经常大声喧哗，利用办公电话到处大声打电话，在办公室里抽烟，而且还经常逞能，走到别人后面得意地告诉别人：“I just found a bug! (我刚刚找到一个 Bug!)”，好像别人都发现不了 Bug。但是，后来我警告过她以后，她就努力改正了这些毛病，并且非常认真敬业。她学得非常快，三个月以后，就已经非常专业了。最后，我的老板终于承认她真是非常厉害，并将她转为了正式职员，现在她恐怕已经成为测试组长了。

从这个例子可以看出，做一名好的普通测试人员并不需要某方面（如计算机）的专业基础，我们不能像评价开发人员一样依照常理来评价测试人员。开发人员肯定需要专业基础，但是测试人员有多种，如做 IE 的一些手工测试（Manual Test）就不一定需要任何专业基础，他们是在以用户的身份使用（甚至乱用）产品并从中发现问题。那位家庭妇女就具有这方面的天分。我当时也招聘了一些学生物、物理专业的博士，但他们做得并不怎么样，远不如这位家庭妇女干得好。





总结

总之，软件测试是一门非常崭新的学科，目前还处于“婴儿”阶段，没有人确切地知道它需要什么样的基础，也没有人确切地知道它应该怎样发展。因此，软件测试学的发展还有赖于大家共同努力，以促进对其研究的不断深入。

第 9 章



软件测试基础
Basic of Software Testing

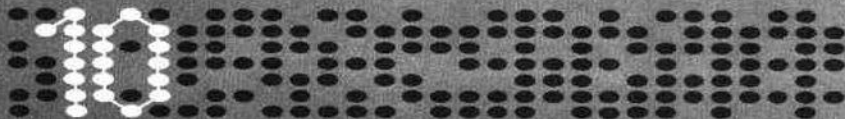
第10章

如何撰写测试文档

Write Good Testing Documents

背景

正因为软件测试还是一门新兴学科，各方面的规范还不完备，目前软件测试工作还无一定的规范可依据。微软经过二十余年的发展，在实践中不断摸索和总结出一套独特的软件测试方法。本章中，陈宏刚博士主要结合自己在微软公司多年的测试工作总结出宝贵的经验，为读者介绍如何撰写测试的相关文档，包括测试计划文档、测试规范文档、测试案例文档、测试报告文档以及Bug报告文档。本章附件中包含了陈宏刚博士亲手撰写的一些测试文档实例。



陈宏刚博士与参加夏令营的同学们在一起



本章内容概览

- 测试计划 (Test Plan)
- 测试规范 (Test Specification)
- 测试案例 (Test Case)
- 测试报告 (Test Report)
- Bug 报告 (Bug Report)

10.1 测试计划 (Test Plan)

测试计划通常可以分为以下两种。

(1) 作为产品的测试计划

用来组织和管理测试工作的文档。这些文档通常是作为产品来开发的。例如，通常为军方开发产品时，由于军方对软件的要求很高，在交付时往往需要附加一个完整的测试计划，就像产品一样。然后，他们就会根据你提供的测试计划一步一步地做，测试软件是否达到了他们要求的性能？是否实现了他们要求的所有功能？其实这种测试计划就是一种产品，所以必须将它写得非常好、非常详细。

(2) 作为工具的测试计划

用来管理测试项目并查找 Bug 的文档。这些文档是对测试工作进行扩展的有用工具。微软的产品是面向一般用户的，所以一般不会 Ship（发行）测试计划，因为我们提供的帮助文件已经足够完备了。

测试计划只是我们内部使用的，所以往往都做成一个工具。

我所介绍的测试计划指的就是我们微软内部使用的一般方式，



即作为工具的测试计划。

在微软，一个项目经理负责一个大的产品领域，该领域又分为许多小的领域，就是由开发人员和测试人员负责的部分。例如，当年我在一个产品团队时，我们的团队由 3 个项目经理、8 个开发人员、15 个测试人员组成，所以一个开发小组（Team）一般是由 1 个项目经理、2~3 个开发人员、4~5 个测试人员组成。这样，一个领域就需要一个测试计划。所以，我们在写自己的测试计划之前，必须有项目经理提供的产品特性（Function）计划文档和开发人员提供的开发进度（Schedule）文档，前者对某一项产品特征给出详尽的描述，后者规定软件应该完成的时间，并且该进度必须要符合整个产品的进度。

前面已经说过，测试是一个新领域，还没有固定的格式和要求。根据每一个测试人员具体任务的不同，每一个测试人员编写的测试计划会有所差异。测试计划和产品开发紧密相关，由多个部分组成。所有大型的商业软件都需要有完整的测试计划，需要具体到每一个步骤，并且每一个部分都必须符合规范要求。

那么，测试计划到底要包括什么呢？这里我用以前做的一个测试计划为例，来介绍一下一份完整、规范的测试计划应该包括的要素。但这个例子，只是给大家起一个参考作用。每个人都有自己的喜好，可以根据自己认为重要的事情编写测试计划。

通常，测试计划应该包括如下内容。

（1）概述（Overview）

在测试计划文档的开始是一个概述（Overview），也就是说，测试计划首先应说明该测试是做什么的。

（2）测试目标和发布标准（Test Goals and Release Criteria）

一般来说，测试计划文档里一定要有测试的最终目标（Test Goals），必须使自己及别人明白为什么必须做这个测试，该测试需要达到的目的是什么。



另外，测试计划要明确定义发布标准（Release Criteria）的范围，并为每一个发布标准定义详细的阶段性目标。表 10.1 是一个发布标准的例子。

（3）计划将测试的领域（Planned Test Areas）

测试计划中要明确给出将要测试的特性领域和主要功能。该计划中列出的特性/功能测试应该覆盖被测试产品的所有特性，包括对话框、菜单和错误信息，以及每个领域的关键功能。同时，测试计划还应该给出对应的每个测试领域的测试规范（Test Specification）文档的路径。

表 10.1 发布标准

Area	RTM Rate	Beta 1	Beta 2
BVT Test	100%	100%	100%
Acceptance Test	100%	95%	100%
Stress	72 h	24 h	48 h
Full Automation Pass	#	#	#
Code Coverage	#	#	#
International Sufficiency	#	N/A	90%

（4）测试方法描述（Testing Approach/Description）

从总体的角度定义产品的测试方法，如前面讲过的 BVT, Automation, Stress, Performance 或 Compatibility。

（5）测试进度表（Testing Schedule）

测试计划需要为测试的每一个阶段定义详细的进度表，并且该进度表必须与项目经理的要求，以及产品开发的进度相一致。实际上，测试进度表依赖于项目经理和开发人员制定的进度表。

（6）测试资源（Testing Resource）

定义参与测试的人员，描述每个测试人员负责的领域，并给出



对应的项目经理和开发人员的信息以及他们的文档路径。表 10.2 是一个测试资源的例子，其中，如果某测试领域已有特定测试人员负责，该领域的负责人 (Owner) 项就是该测试人员的名字，否则以 TBD (to be decided, 待分配) 或 TBH (to be hired, 待雇佣) 代替。

表 10.2 测试资源

测试领域	负责人
XML Update Grams	George
OpenXML	Art
XML Query	Hoang
ISAPI	Howard
XML for OLE DB	TBD
Security	TBH
International Sufficiency	TBD

(7) 配置范围和测试工具 (Configuration Coverage and Test Tools)

测试计划中必须给出测试所需的机器平台及相关硬件配置。例如，我们公司的测试计划中就考虑到了各种平台和机器硬件的组合。每一个计算机硬件公司都派代表驻在我们微软总部，主动把机器送来供我们测试。因为他们希望我们的产品都能在他们的机器上运行。但有趣的是，往往我们的产品在某一个公司的机器上运行得很好，但在另一个公司的机器上运行得不好，这时有两种可能的情况：一种是我们产品的兼容性不好，另一种情况就是该公司的硬件有问题。因为他们都是将新机器拿来给我们测试的，而且与软件一样，硬件同样也存在 Bug，所以如果出现上述问题，并且是后一种情况，这时，他们就会将机器运回公司，重新进行修改。





例如，有一次我在某品牌笔记本电脑上安装了我们的一
个新产品，可就是安装不上，而在其他公司（如 Dell, IBM 等）
的笔记本电脑上都能正确安装。由于我们实在找不出软件方
面的问题，于是就把该公司的人员（他们也驻在我们公司总
部）找来，他们看了以后，也搞不明白，就将机器送回他们
公司进行检测。结果发现有一个硬件问题，更正之后，我们
的产品就能正确地安装了。因此，如果出现软件安装不了的
情况，不一定是软件的问题，也有可能是硬件的问题。

另外，测试计划中还必须说明将使用的测试工具。测试工具非
常重要，我们可以利用已有的工具，如果没有合适的，还必须自己
开发。所以，测试人员在测试过程中也有可能需要编制程序。通常，
一个大的测试团队里都有一个小团队专门管理并开发测试工具。这
就需要这些测试人员具有很强的编写代码的技能和计算机专业知识
基础，这些测试人员通常被称为 SDE/T (Software Design Engineer in
Test)，他们需要具有与开发人员一样的能力，但他们主要是编制一
些测试工具。其实，开发测试工具是需要很多时间的，所以微软在
此方面投入了很多的人力。

为了使大家能够对测试计划的撰写有一个更加直接、更加具体
的印象，本书第 304 页附件 10.1 中给出一个真实的测试计划，供大
家学习参考。这是编者在 SQL Server 的 XML 团队时写的一个测试
计划。



10.2 测试规范（Test Specification）

在编写测试规范之前，需要参考项目经理写的产品规范，以及开发人员写的开发计划。但每个开发人员的习惯都不一样，有的习惯写开发计划，这样我们写测试规范和测试计划就很容易了，有的开发人员只是让你参照项目经理的规范，这时，我们就需要费点精力。另外，每一个领域都应该有一份详细的测试规范，所以还需要参照测试计划，测试规范的内容要与该计划相吻合。

下面介绍一下测试规范文档应该包含的要素。

（1）背景信息（Background Information）

背景信息包括：项目经理写的规范所在路径（如表 10.3 所示）；负责产品的人员，包括项目经理、开发人员和测试人员（如表 10.4 所示）；产品修改记录（如表 10.5 所示）。

表 10.3 参 考 规 范

Name of Document	Revision	Author
Message Tracking Center Specification(Web linked)	10/17/99	Barry

表 10.4 相 关 人 员

Name	Title/Involvement with Area
Barry	Program Manager
Michio	Developer
David	Developer
George	Develop—MAD.exe
Alex	Transport Component Development
Milan	Transport Component Development
Steve	MTA Component Development
Steven	Admin Test Manager
Victoria	Admin Test



表 10.5 修改记录

Date	Name	Comments
12/11/98	Lei (Dexter)	Initial Draft
10/15/99	Victoria	Revised test specification

(2) 被测试的特性 (Features to be tested)

被测试的特性包括：单个特性、一个领域内的组合特性、与其他领域中的特性相集成的特性以及没有覆盖到的特性。

在测试时，虽然有时需要使用某特性，但它可能并不属于你所负责的领域，这时，将它标志为没有覆盖到的特性即可。这样，能让别人清楚地知道，你已做的事件和未做的事情。同时，它还可以帮助新来的人了解他应该做的事情。

(3) 功能考虑 (Function Considerations)

测试规范中应该提供详细的功能描述，包括菜单、热键、对话框、错误信息和帮助文档。

(4) 测试考虑 (Test Considerations)

一般要有测试假设，这是最基本的测试考虑。项目经理写的规范一定要包括我们要做的所有事情。如果项目经理更新了他的规范，那么我们也必须相应地修改我们的内容。另外，还要考虑各种边界情况、不同的语言所使用的符号、系统测试和黑盒测试。

(5) 测试想定 (Test Scenarios)

测试规范最重要的内容就是阐述具体的测试方法。对每一个特性或功能，给定一些测试想定。根据测试想定，我们可以很容易地产生测试案例。表 10.6 是一个测试想定的例子。

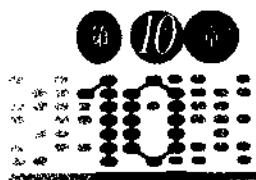




表 10.6 测试想定

Test Case ID	Priority	Tactics ID	Test Description	Test Structure	Expected
1	0	1203	Insert a row with no updg:before block	Basic format without updg:before block	The row is inserted
2	1	40	Insert a single row with one given value and empty "before?block in a table without requiring 2+specific column values	Use the basic format.Table allows nulls for columns not specified	The row is added into the bottom of the table
3	1	41	Insert multiple rows in a same table with multiple given column values with empty "before?block	Multiple rows appear inside of the updg:after block	Those rows are added into the bottom of the table with the given values specified
4	1	42	Insert multiple rows in different tables in one updg:after block	Similar to Case 41 with different table names in related rows	Those rows are added into the bottoms of the table with the given values specified
5	1	43	Insert a single row with multiple given column values without "before?block	Basic format without updg:before block	The row is added into the bottom of the table with the given values
6	1	44	Insert a single row with a sql:id setting in updg:before block but not in updg:after block	Basic format with nonempty updg:before block and the inserted row is in updg:after block	The row is added into the bottom of the table

有时，因为必须考虑到所有的情况，所以一个测试想定可以写多个测试案例。表 10.6 中，一个测试想定是对应于一个测试案例的。其中，Tactics 是我们自己设计和编写的一个工具，它的功能是直接将测试案例放到数据库中，可以用 Web 直接浏览的。

本书第 312 页附件 10.2 所示的是与附件 10.1 中所示的测试计划相对应的一个测试规范。这也是作者写的一个真实的测试规范。

其实，测试规范不止有一种撰写方法。本书第 318 页附件 10.3 所示的就是一个更生动的图文并茂的测试规范。





10.3 测试案例 (Test Case)

在开发测试案例之前，必须具备一份正确的项目经理规范和一份详细的测试规范。在开发测试案例时，最初的案例一般是根据规范中的定义而开发的。在运行的过程中，根据测试反馈信息，我们可能会发现未考虑到的新问题，这就需要不断地添加测试案例。另外，当我们发现新的 Bug 时，也可能需要添加新的测试案例，这样，就可以极大减少回归测试。

测试案例没有固定格式，只要清楚表明了你的步骤和需要验证的事实，使得任何一位用户或工程师都可根据你的测试案例的描述来完成该测试。本书第 330 页附件 10.4 中所示的是两个测试案例实例。

10.4 测试报告 (Test Report)

在测试的过程中，测试管理人员会定期汇报测试进展。通常，测试管理人员以测试报告的形式向整个产品开发部门报告测试结果及所发现的 Bug。撰写测试报告的目的就是为了让整个产品开发部门了解产品开发的进展情况，以及使 Bug 能够迅速得到修复。

测试报告的格式并无定式，测试管理人员为不同的产品所规定的测试报告可能都不相同。只要所写的测试报告能够完整、清楚地反映出当前的测试进展情况即可。最重要的一点是测试报告要易懂，不要让人迷惑或产生误解。

附件 10.5 所示是我在 IE4 团队时所写的一个测试报告，主要是对 IE4 站点访问功能的测试结果作了一个总结。



10.5 Bug 报告 (Bug Report)

一份 Bug 报告应该包括以下几个基本要点:

(1) Bug 名称 (Bug Subject/title)

需要用一句简单的话来描述 Bug 的主要问题, 这样, 开发人员就可以知道出现了什么样的 Bug。

告诉别人测试使用的机器、操作系统以及软件平台。

(2) 被测试软件的版本 (Software Version)

指出测试人员测试时的被测软件版本, 开发人员在纠错时才有针对性。

(3) 优先度与严重性 (Priority and Severity)

指出 Bug 的优先度和严重性程度。Priority 和 Severity 二者是不同的, 前者是针对项目经理定义的特性, 而后者是针对使用方的, 即对用户来说, 该 Bug 的严重性。

(4) 报告测试的步骤 (Report Steps of Test)

在这里, 说明 Bug 是在什么样的操作步骤之后产生的。例如, 先打开一个文件夹, 再……然后 Bug 出现。这些内容可以由测试工具自动产生。

(5) Bug 造成的后果 (Result)

说明在实际操作时, 该 Bug 造成的后果。

(6) 预计的操作结果 (Expect result)

给出测试人员认为应该出现的结果。



(7) 其他信息 (Other Information)

比如, 这个 Bug 在某些情况下 (如正在打开另一个文件夹等的时候) 并不存在, 或是在老的版本中没有出现等。凡是测试人员认为有助于修正 Bug 的信息, 都可以在这里提及。

例如, 附件 10.6 中所示的就是一个简单的 Bug 报告。

有时候, Bug 报告是非常复杂的。测试人员和开发人员之间会多次进行反复“对抗”。例如, 附件 10.7 中所示的就是一个比较复杂的 Bug 报告。

这个 Bug 报告进行了多个回合的“对抗”。在修复 Bug7187 之前, 测试人员和多个开发人员进行了反复“对抗”: 开发人员认为 Bug 已经修复, 但测试人员又重新激活了该 Bug, 于是开发人员又回过头进行修复, 直到最后把该 Bug 真正修复。

所有这些 Bug 报告都放在一个数据库中, 它为团队中的所有人提供了相当重要的信息。团队中的项目经理和决策人员根据这些 Bug 的统计数字和走势了解产品的进度。开发人员可以通过这些 Bug 报告中清晰的测试步骤比较容易地找到问题, 修复好 Bug 后, 又会将修复的信息写到 Bug 报告中, 供测试人员检查 Bug 是否真正地被修复。如果测试人员发现问题仍没有解决, 他将重新激活 Bug 报告, 输入新的信息, 提供给开发人员。

由此可见, Bug 报告在产品的开发中占有核心地位。能够写出清晰明确的 Bug 报告是对一个测试人员最基本的要求。

附件 10.1 SQL Server XML Shiloh 测试计划

SQL Server XML Shiloh Test Plan

Owner: George Chen

Last Modified: 04/30/1999 16:12:34

0. Overview

This test plan describes how SQL Server XML is tested during Shiloh development, in order to make the product as robust as possible, from a high-level of views. The overall objective for us is to ...

SQL Server XML testing consists of both automated and ad hoc tests. This milestone's focus is building a structured suite of tests for new SQL Server XML features, based on test specs and test spec reviews, to compliment ad hoc testing.

Automated test tool development, test case writing, and test case maintenance occurs at the beginning of the milestone, mixed with ad hoc days. Regular automated tests, and ad hoc testing done during the test case writing stage catch a large percentage of bugs in the beginning of the milestone. As the milestone nears completion, regression test passes are run.

1. Shiloh XML Test Goals and Release Criteria

Test Goals

- To ensure the new XML features and functions in Shiloh work as expected.
- To cover all of SQL Server component areas.
- To focus on API related client testing.

- To do code coverage tests using existing tools/clients.
- To run reasonable time of stress and performance testing.

Release Criteria

Area	RTM Rate	Beta 1	Beta 2
BVT Test	100%	100%	100%
Acceptance Test	100%	95%	100%
Stress	72 hours	24 hours	48 hours
Full Automation Pass	#%	#%	#%
Code Coverage	#%	#%	#%
International Sufficiency	#%	N/A	#%

2. Testing Approach/Methodology

General Approach

- Use IE5, IE4.01 SP1 to access the SQL Server.
- Use Netscape to test non-XML returning scenarios.
- Write own clients/applications to do the test if there are not related existing tools.
- Use existing tools/CmdLine to run performance and code coverage tests.
- Run automated p1 BVT tests daily by the build lab.
- Run automated acceptance test cases after the BVT passes and the new build is released.
- Run automated other tests at least once a week by the build lab.
- Run stress and performance tests regularly.
- Do formal regression test passes near the end of the milestone.

第 10 章



Result Comparisons

We will set standard result files for test result comparisons. These files will be created and maintained either manually or automatically and will be stored in special machines that are convenient to do the comparison with the testing results. These files are especially important for our test automation.

Real-world Scenario Approach

Considering the needs of the real-world users and customers, we will use Web clients or write user-oriented application/cases to do many of our tests.

Code Coverage Scenario Approach

We will use existing tools like CmdLine and Query Analyzer to do some of our tests in order to do a better code coverage test.

Basic Verification Test (BVT)

The purpose of our BVT test is to make sure the very basic and simple new feature cases function as we expect. We will integrate our BVT test into SQL Server BVT Lab and run it following the lab schedule. The BVT test cases are created through the DRT cases of our developers and by our testers. The BVT test will be run on different system configurations depending on different components.

Build Acceptance Test (BAT)

We will do a build acceptance test after the BVT is passed for the build. The BAT test cases are created by our testers and reviewed with related PMs and Devs. The



purpose of BAT test is to make sure the build is acceptable to give to people to test or try it. The BAT test will also be run on different system configurations depending on different components.

Test Case Implementation

- All of our test cases should be automated.
- We should have either existing or new developed tools to run the test cases.
- We should compare our test case results with a master database results.
- The test cases will be run on different system configurations depending on different components.

HTTP/Internet Access to SQL Server

We shall dedicate our major effort to do user API testing and give our feedback to the PM and Dev.

...

Extended OLE DB Access to SQL Server

We shall also dedicate similar effort to do tests through OLE DB clients.

...

Stress Testing

We will do our stress test in both our test lab and run it on our office machines regularly. The stress test cases will come from our other test cases. The tool to run the stress test will probably be the "Homer".

Performance Testing

The performance test will be run in our test lab mainly. It will use Web related clients, self-developed clients

and existing tools.

3. Planned Test Areas

ISAPI

This area will be tested mainly through IIS from HTTP clients. ISAPI will be the extension/filter for the data from the client. The client data will include ... and ... methods. We need to test both valid and invalid data. This is also a major concerned area for security. Involved servers for this area will include NT Server, IIS, SQL Server. The test specification for this area could be found in XML ISAPI Specification.

XML Query

This area consists of XML Query output and Returning XML from TSQL. There are 3 access methods for this area:

- (1st Method)
- (2nd Method)
- (3rd Method)

Thus, this will be tested through ... or ... or ... Besides developing our own test specs and cases, we also can leverage other related teams documents and test cases. The test specification for this area could be found in XML Query Specification.

XML Update grams

Unlike XML Query, this area has only 2 access methods and will be tested through either ... test or self-developed ... clients. We can't use ... access for this area. The test specification for this area could be found in XML Update grams Specification.



Managing XML with Stored Procedures

The goal of this feature is to provide a mapping of ... or ... data into relational tables. Similar to XML Query, this area has 3 same access methods. Hence, this area will be tested through (1st Method), or (2nd Method), or (3rd Method). The test specification for this area could be found in OpenXML Specification.

Security

Security issues appear in almost every area and need to be considered seriously. This is very important from the user's view. We need to have a special effort to test the security in our components. However, more details need to be defined for this area before we could go further to write our specification.

International SQL Server XML Testing

Sufficiency Testing

The SQL XML Team creates, maintains, and performs international sufficiency tests for SQL Server XML. This means testing XML support of SQL Server with DBCS, Unicode, and general extended character support. Testing of these areas with US builds of SQL Server, is performed on Pan-European and Japanese operating systems.

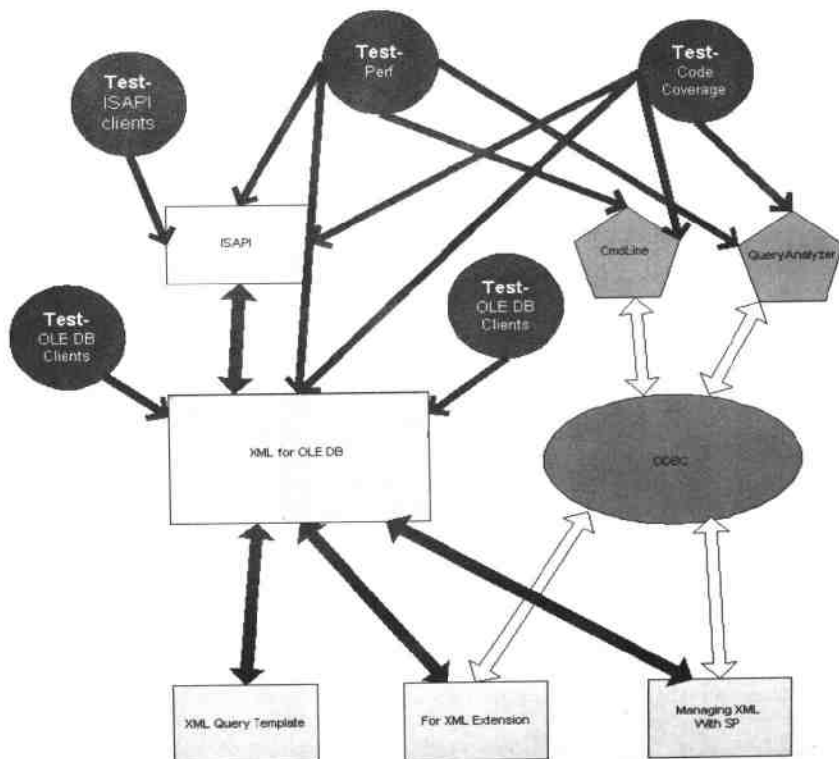
General International Testing Issues, Beyond Sufficiency Areas of SQL Server XML that need specific international testing are to be determined on a case-by-case basis. We will work together with our PM team, Dev team and

SQL Server International Test Team to ensure no duplication of effort exists and good coverage, along with maximum leverage of our team's expertise.

For areas where both test teams have ownership, test case breakouts are to be held where both teams meet to discuss how these areas are tested, what cases are written, and who owns what specific test cases.

It is important to note that a test case that is best tested by the International test team is not necessarily owned completely by that team.

4. Test Architecture





5. Test Resource

The testing ownership for each area is in the following:

Test Area	Owner
XML Update Grams	George
OpenXML	Art
XML Query	Hoang
ISAPI	Howard
XML for OLE DB	TBD
Security	TBH
International Sufficiency	TBD

6. Coverage and Test Tools

- We'll consider all SQL Server editions. But our focus will be on ... and ... editions.
- Windows 95, Windows 98, Windows 2000, Windows NT 4.0 Server, SBS, Enterprise Server Edition, Workstation
- Intel and Alpha machines.
- Tactics as test case management tool.
- Harness and Robovision are going to be the automation tools.

7. Schedule

The complete schedule is being developed. As soon as it's available, it can be found here.

附件 10.2 XML Update Grams 测试规范

XML Update Grams Specification

Last Updated: (06/8/1999)

Author: George Chen

Status: Draft

1.0 Description/Purpose

This describes the test side of the XML Update grams as SQL Server extensions. Update grams are one part of the Shiloh implementing XML support within SQL Server to allow for data exchange across the Internet. This feature is useful for customers to send singleton row modifications to the database via XML and is also flexible. The corresponding PM specification could be found in Matt Domo's Functoinal Spec.

2.0 Approach Refinements

2.1 Cases not tested

- Access through ...
- ... OS platforms.
- Those not covered in the corresponding PM's spec.

2.2 Testing Assumptions

- The corresponding PM's spec covers the necessary feature of XML Update grams. When the PM's spec changes, this spec needs to be updated too.

2.3 General Test Approach

- Use ... to access the XML Update grams extensions.
- Use ... through ... to call the XML Update grams.
- Use ... through ... to connect the XML Update grams.

- Test the XML Update grams with all of SQL Server editions.
- Run test on major ... OS platforms.
- Test with the intention to catch as many bugs as possible through user and code coverage scenarios.
- Get test cases into a stage where anyone can run them so we don't need an expert to see if they pass.

2.4 Testing Considerations

- Keywords function as expected.
- Only the XML Update grams formats can be run correctly.
- The XML Update grams could combine with other parts of SQL Server XML like ... successfully.
- The localizations need to be checked if we have related international builds.
- The error handling should be considered.
- Stress and Perf should be considered too.

3.0 Requirements

...

4.0 Breakout

4.1 Individual Tags

- ...

4.2 Basic Format with NameSpace

The basic format with a namespace '...' for the XML update grams will be in the following:

```
<...:sync xmlns:sql="urn:schemas-microsoft-com:xml-sql">
  <sql:before>
    <TABNAME [sql:id="value"]
      col=[@identity|@@identity]
```

```
col="value" col='value' .../>
</sql:before>

<sql:after>
  <TABNAME      [sql:id="value"]
  col=[@identity|@@identity]
  col="value" col='value' .../>
</sql:after>
</sql:sync>
```

Here TABNAME is the name of the table the operation will be performed against.

4.3 Special Cases

See Section 5.0.

5.0 Test Identification

5.1 Insert Format

The basic insert format with a namespace 'sql' for the XML update grams will be in the following:

```
<...:sync>
  <...:before>
  </...:before>

  <...:after>
    <TABNAME pKey1="PrimaryKeyValue"
    pKey2="PrimaryKeyValue2"
    colA="NewValueofColA"
    colB="NewValueofColB" .../>
  </...:after>
</...:sync>
```

The above operation will insert a row into the related database.

Test Case:	Priority:	Test Description:	Test Structure:	Expected:
1	1	Insert a single row with one given value and empty "before" block in a table without requiring 2+ specific column values	Use the basic format.	The row is added into the bottom of the table if the value can be specified
2	2	Insert a single row with multiple given column values with empty "before" block	Use the basic format	The row is added into the bottom of the table with the given values specified
3	1	Insert multiple rows in a same table with multiple given column values with empty "before" block	Multiple rows appear inside of the ...after block	Those rows are added into the bottom of the table with the given values specified
4	2	Insert a single row with a given identity ID value with empty "before" block	Use the basic format with pKey="GivenValue"	Error
5	1	Insert multiple rows in different tables in one ...after block	Similar to Case 3 with different table names in related rows	Those rows are added into the bottoms of the different tables with given column values with empty "before" block
6	2	Insert multiple rows in different tables in multiple ...after blocks	Multiple ...after blocks appear	Error: A name was started with an invalid character
7	1	Insert a single row with multiple given column values without "before" block	Basic format without ...before block	The row is added into the bottom of the table with the given values specified
8	1	Insert a single row with a same ...id setting in both ...after and ...before blocks	Basic format with non-empty ...before block and the inserted row is below or before the identified row	The row is added into the bottom of the table

Test Case:	Priority:	Test Description:	Test Structure:	Expected:
9	1	Insert a single row with a ...id setting in ...before block but not in ...after block	Basic format with nonempty ...before block and the inserted row is in ...after block	The row is added into the bottom of the table
10	2	Insert a row which already exists in the table without given ID column value	Basic format	The row is added into the bottom of the table
11	2	Insert a row which already exists in the table with partial values specified and empty "before" block without given ID column value	Basic format	The row is added into the bottom of the table with some value NULL
12	2	Insert a single row with required non-ID primary key value and without given the primary key value	standard format without the pKey value	Error
13	2	Insert a single row with the ID primary key and trying to give ID key value	Standard format	Error
14	1	Insert a single row with the ID primary key and try to given ID key value with ...@identity method	<Tabname ...:at-identity="x"/><Tabname pKey="x" .../>	A row is inserted successfully
15	1	Without the ...:sync tag	Without <...:sync> and </...:sync> in the script	Error
16	2	Insert a single row with one given value and empty "before" block in a table requiring 2+ specific column values	Use the basic format	Error
17	2	Insert a row before a unchangeable row under table constraints without shl:id setting	...:before is not empty	Error
18	1	Insert a single row with ...id setting with non-empty ...:before block	Basic format with non-empty ...:before block and ...id in the identity row	The row is added into the bottom of the table

Test Case:	Priority:	Test Description:	Test Structure:	Expected:
19	2	Insert a single row with <code>....query</code> tag inside of a <code>....sync</code> tag	Basic format with the <code>....sync</code> tag including a <code>....query</code> tag	Error
20	2	Insert a single row with a capital <code>....SYNC</code> tag	Basic format with the <code>....SYNC</code> tag	Can't insert and no error. Just the update scripts shown
21	2	Insert a single row with a capital <code>....AFTER</code> tag	Basic format with the <code>....AFTER</code> tag	Can't insert and an error is given
22	3	Insert a single row with a wrong <code>mlns</code> of <code>xmlns</code> inside the <code>ROOT</code> tag	Basic format with the <code>mlns</code> instead of <code>xmlns</code> inside of <code>ROOT</code> tag	Can't insert and an error is given
23	3	Insert a single row with <code>/....after</code> misprinted as <code>....after</code>	Basic format with the <code>mlns</code> instead of <code>xmlns</code> inside of <code>ROOT</code> tag	Can't insert and an error is given
24	3	Try to insert a row in the following script without <code>ROOT</code> tag	Basic format without <code>ROOT</code> tag	Can't insert and an error is given



Total Test Cases = 18

5.2 Delete Format

... (省略)

5.3 Update Format

... (省略)

5.4 Mixed Format

... (省略)

6.0 Cases from Other Places

Not applicable.



7.0 What's New

Name:	Date:	Changes:
George Chen	5/25/1999	Create the first draft 0.1
George Chen	5/28/1999	Modified to get the second draft 0.2
George Chen	6/2/1999	Added the test cases for the third draft 0.3
George Chen	6/8/1999	Modified the test cases for the first version 1.0

附件 10.3 另一种格式的测试规范

Virtual Directory User Interface Testing

SQLVDir ... can be used to configure a ... With this tool, you can create an IIS server ... to connect the IIS server and the SQL server. Globalization testing team needs to make sure that localizable items are able to take locale characters as parameters.

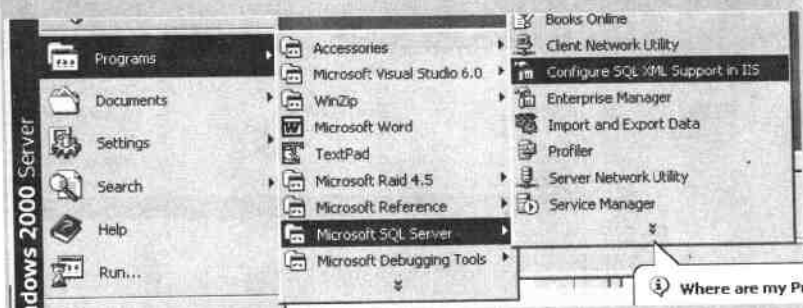
1. Configuration Setup

- A SQL server running on a localized OS M1 (NT 4.0 or NT5.0)
- An IIS server running on a localized OS M2 (NT4.0 or NT5.0) that has SQL client and server tools installed.

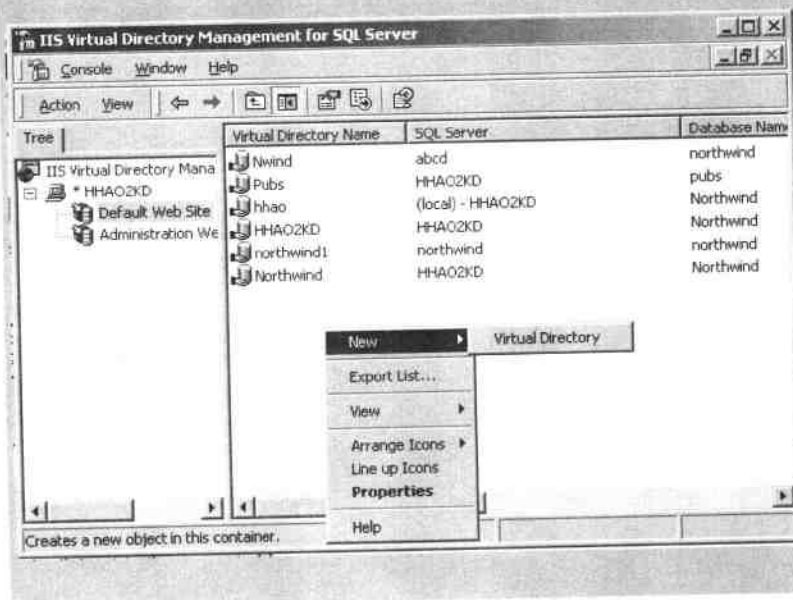
2. Test case execution

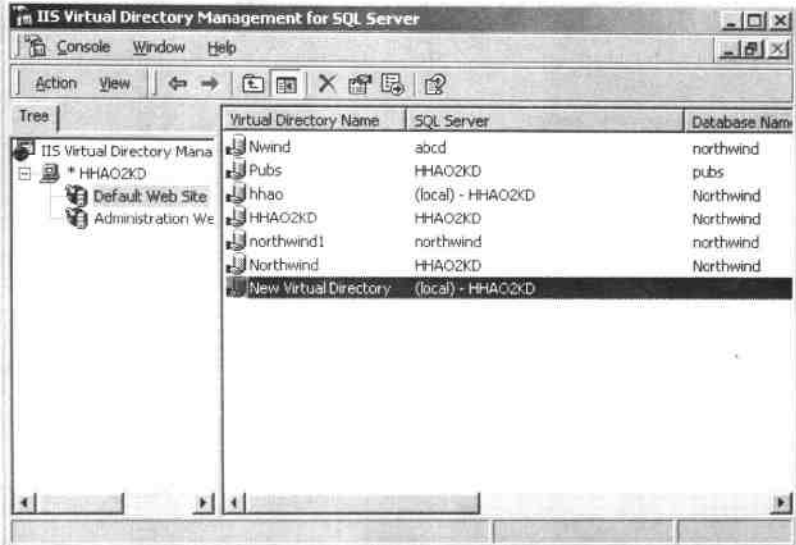
The SQL VDir tool can be executed on a remote machine. To simplify our test case scenarios, we assume the tool is only executing on the IIS machine M2.

2.1 Launch the program



2.2 Create a new virtual directory





2.3 Enter the Virtual directory's Parameters

Double click the "New Virtual Directory" entry. The following properties dialog box will show up. We will walk through every property tag to entry characters with local machine code page.

2.3.1 General

Northwind Properties

General | Security | Data Source | Settings | Advanced | Virtual Names

Virtual Directory Name

The name through which users will access the database. For example, if you specify "Northwind", users will be able to access the database by navigating to: "http://myserver/Northwind"

Local Path

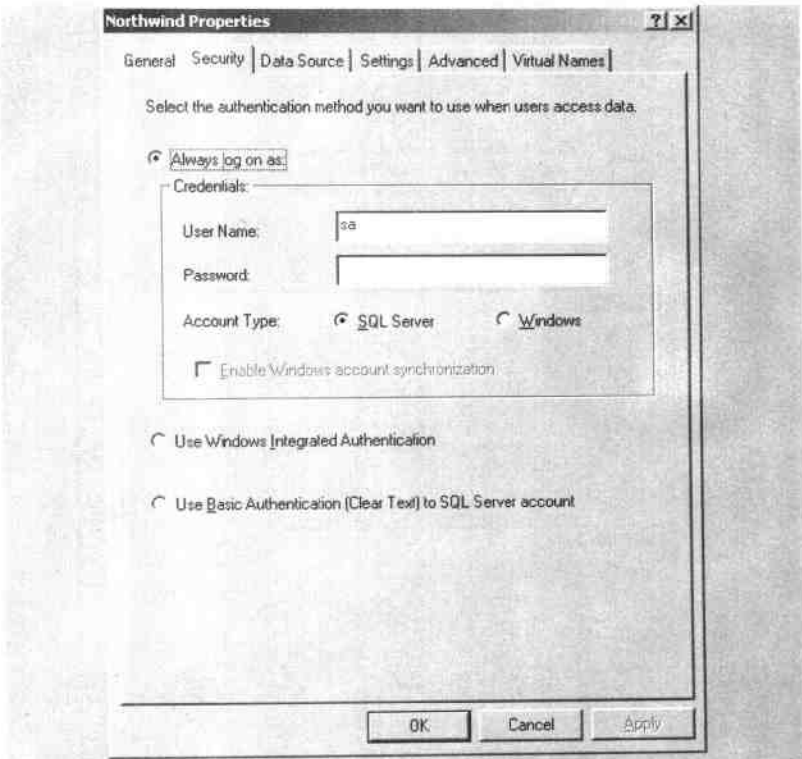
The full path to the actual directory that contains the files you want to make accessible through this virtual directory.

Item Name	Type	Take Localized Characters	Value
...	Edit Box	Yes	
...	Edit Box	Yes	

2.3.2 Security

Globalization testing in this area only involved in SQL server authentication modes. Please make sure that you have created a user account with SQL server that contains the localized characters.





Select Always log on as radio button:

Item Name	Type	Take Localized Characters	Value
...	Edit Box	Yes	
...	Edit Box	No	


2.3.3. Data Source

Please make sure that the SQL Server (M1) name contains the localized characters. Also the SQL Server has a database with localized database name.

Northwind Properties [?] [X]


General | Security | Data Source | Settings | Advanced | Virtual Names

SQL Server

 The name of the server that stores the data you want to publish through this virtual directory. Optionally, the instance of SQL Server running on the specified server. Click the button to browse for a server.

HH&02KD [Browse]

Database

 The name of the default database on the SQL Server specified above. Click the down arrow to connect to the server with the credentials specified on the security tab and retrieve the list of database names.

Northwind [v]

[OK] [Cancel] [Apply]

Item Name	Type	Take Localized Characters	Value
...	Edit Box	Yes	
...	Combo Box	Yes	

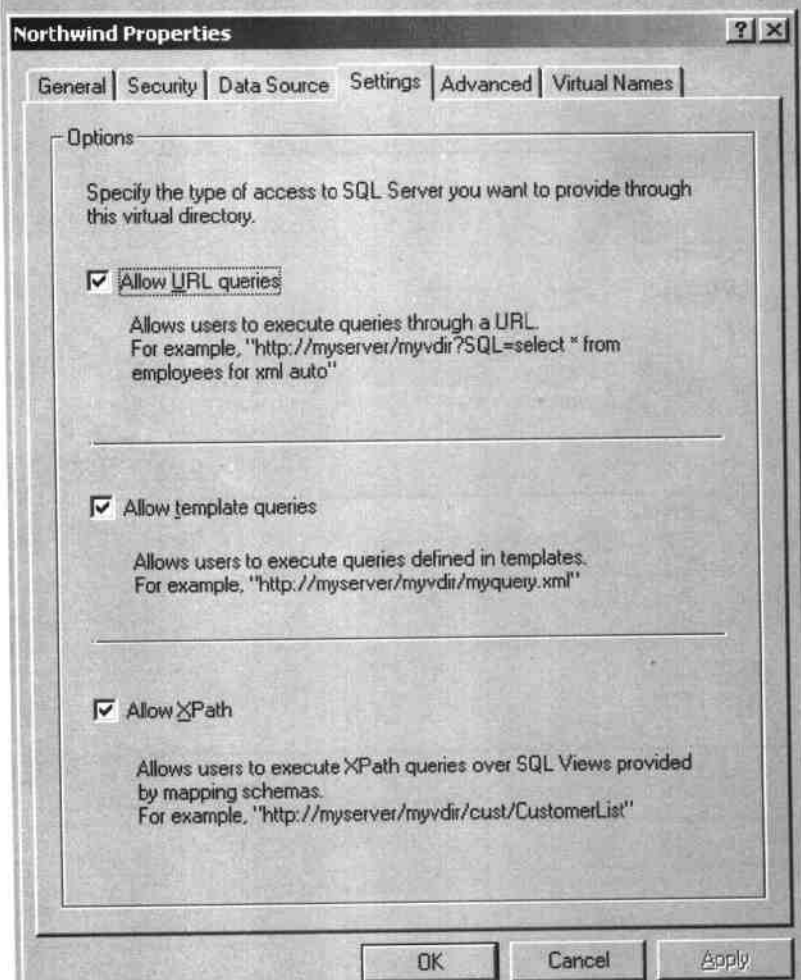
You should be able to see all the database names in the Database combo box.

第 10 章

Write Good Testing Documents
如何撰写测试文档

2.3.4 Settings

Select all the settings.



2.3.5 Advanced

Northwind Properties

?

×

General

Security

Data Source

Settings

Advanced

Virtual Names

ISAPI Location

Set the location of the SQLISAPI.DLL required for accessing data through this virtual directory.

ram Files\Common Files\System\OLE DB\sqlisapi.dll

Browse...

OK

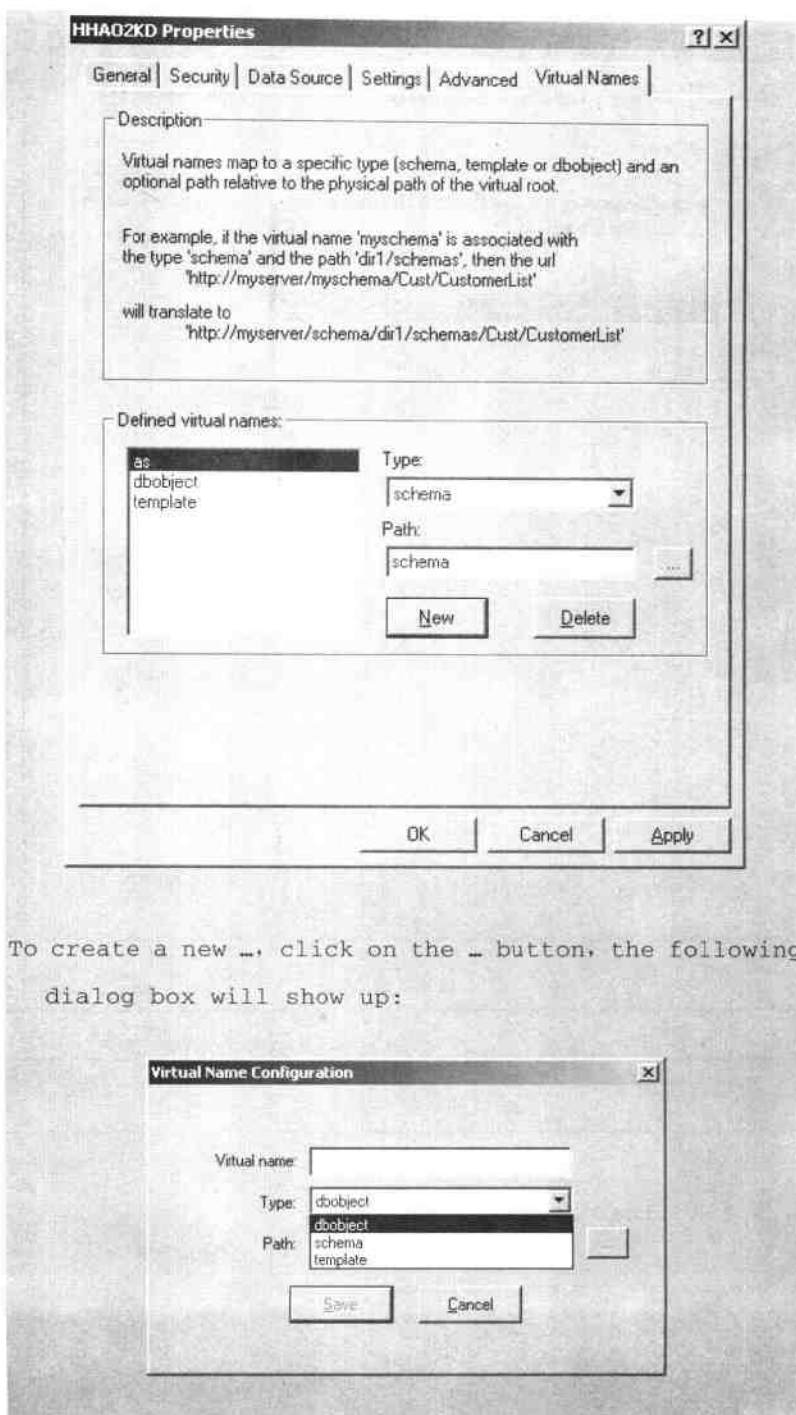
Cancel

Apply

Item Name	Type	Take Localized Characters	Value
...	Edit Box	Yes	

2.3.6 Virtual Names





Create three ...:

2.3.6.1 DBOBJECT

Item Name	Type	Take Localized Characters	Value
...	Edit Box	Yes	
...	Combo Box	No	...
...	Edit	Yes	

2.3.6.2 schema

Item Name	Type	Take Localized Characters	Value
...	Edit Box	Yes	
...	Combo Box	No	...
...	Edit	Yes	

2.3.6.2 template

Item Name	Type	Take Localized Characters	Value
...	Edit Box	Yes	
...	Combo Box	No	...
...	Edit	Yes	

Please make sure that the physical path does exist before click on the OK button.

2.4 Click on the OK button

Click on the OK button to save the ... information that has been entered.

2.4 Create a second new virtual directory

Go back to step 2.2 to create another virtual directory, but in **2.3.2 Security** step, select

Use Basic Authentication (Clear Text) to SQL Server Account **radio button** instead.

3. Test Case Validation

To verify if the first virtual roots was created properly, run following four queries in IE5.0 on a client machine. These queries all start with:

http://...

Where ... is the IIS machine name that your ... is created on, ... is the value of ... which you have entered in Step

2.3.1 General

3.1 First ...

3.1.1 SQL= Query

...

- The table can be any SQL database table name that is stored in database that you have entered in step

2.3.3. Data Source.

You should be able to see a XML file returned as query result.

3.1.2 DBOBJECT Query

...

- `_` is the `_` value entered in step 2.3.6.1 DBOBJECT
- The table can be any SQL database table name that is stored in database that you have entered in step 2.3.3. Data Source.
- The column is the column name in the table described above.

3.1.3 Template Query

222

3.1.4 XPath Query

http://...

0000

3.2 Second ...

222

The table can be any SQL database table name that is stored in database that you have entered in step

999

4. Document History

Author	Changes	Date
Howard	Create the doc	01/01/0000



附件 10.4 测试案例实例

案例 1

Title:

Test Microsoft Website Main Hyperlinks Function as
Expected by IE 4.0

Steps:

1. Open IE 4.0
2. Type <http://www.microsoft.com> into Address textbox,
then press Enter button
3. Click on hyperlinks randomly

Verification:

All hyperlinks browse correctly. No any error and
irregular delays

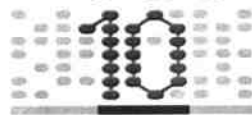
案例 2

Title:

Windows 98 prints DBCS txt files correctly on HP LaserJet
printers

Steps:

1. Turn on a machine with Windows 98 as OS system
2. Select Start/Settings/Printers from the left bottom
corner of Windows
3. Click Add Printer button and add a HP LaserJet printer
4. Highlight the new added HP printer and select File/Set
as Default Printer in Printers Window



5. Close Printers Window

6. Create a new text file with DBCS characters

7. Select File/Print... menu in the text file

8. Click OK button to print the txt file

Verification:

The DBCS file is printed clearly and correctly

Repeat Steps 3 to 8 and the Verification for another HP LaserJet printer until all of HP LaserJet printers are finished.

Note:

One needs to have all of the necessary hardware and software available for the test

附件 10.5 对 IE4 的访问站点功能的测试报告

IE4 RTW Site Test Result Summary

Last Modified: 10/25/1997 16:27:08

1. Test results for top listed sites for IE40

- Currently, there are 111 sites in the list.
- For site information, please refer to IE40 PP2 Site Testing Team Test Plan
- For the detailed IE4.0 test results, please refer to IE 4.0 Top Site Test Results

- For the detailed IE4.01 test results, please refer to IE401 Top Site Test Results.

The summary results currently are:

- Most Category 1 & 2 sites were tested through a modem connection.
- The site pass rate is 100%: all 111 sites are passed for both IE40 RTW and IE4.01.
- Total about ## related pages had been tested/ retested and
- Total about ## related pages have been tested/ retested and ...
- The page pass rate was about 99% for both IE40 RTW and IE4.01.

Pass rate charts

The Site Pass and Fail rates:



The Page Pass and Fail rates:

Results for Page Pass/Fail



(Updated on 10/24/97)

2. MS intranet sites and their results

The summary results for IE40 are:

- Currently, there are ## MS intranet sites or sub-sites in the list for IE4.01.
- For the detailed test results, please refer to MS Intranet Site Test Results.
- Total ## related pages have been tested and no pages of them are failed for IE40 before the RTW.
- The page pass rate was 100% before the RTW.
- The site pass rate was 100%: ## sites/sub-sites are passed and none of them failed.
- Total ## related pages have been tested and no pages of them are failed for IE4.01 after the RTW.
- The page pass rate is 100% for IE4.01.

(Updated on 10/24/97 by GChen)

3. Ad_hoc test sites and their results

The summary results currently are:

- Before the IE40 RTW, there were ## sites in the list.
- For the detailed test results, please refer to Ad_hoc Site Test Results
- Some sites were tested through a modem connection.





- The site pass rate was 100%: all sites were passed and none of them failed before the RTW.
 - Total ## related pages have been tested and no pages of them were failed to pass.
 - The page pass rate was 100% for IE40 RTW.
 - Total # related pages in # sites have been tested and 0 pages of them were failed to pass for IE4.01.
 - The page pass rate was 100% for IE4.01.
- (Updated on 10/24/97)

4. Project Test Results for Site Testing Team

The explanations for the columns in the table:

- Build** -- The used IE40 build for testing
- Project** -- The project assigned to help testers to do Ad_hoc test
- Tester** -- The tester who did the test for the project
- Page # covered** -- The total page number tested for the project
- Site # Visited** -- The total site number visited for the project
- P.P.R.** -- The page pass rate for the project

Project Test Result Summary Through Real-World Browsing:

The following table includes the result summary from the project testing for covering different area and sites. If you want to know the detail testing results for each project, just click the corresponding linked project name.

Build	Project	Tester	Page # Covered	Site # Visited	# Page Failed	P.P.R.	Bug # Raided	Comments
1023	Vacation in Mexico	Bill	54	14	0	100%	0	No bugs were found
1025, 1028	(具体内容省略)							
1028								
1017- 1712								
1028								
1029								
1029								
1029, 1030								
1029, 1030								
1030								
1108								
1108								
1109								
1108, 1112								
1114								
1114								
1114								
1116, 1120								
1116								
1116, 1120								
1116, 1120								
1119, 1120								
1705								
Total	Project Summary	GChen	6725	746	15	99.8%	21	All bugs are fixed now

(Updated by GChen on 10/23/97)

5. Active Bugs and Beta 2 Summary Results

For sitecompat bugs, [click here to see all active sitecompat bugs.](#)

For Beta 2 summary result reports, click the following link:

第 10 章

如何撰写测试文档
Write Good Testing Documents

Beta 2 Test Results.

附件 10.6 一个简单的 Bug 报告

7306 Bug: 'Only in 640x480 resolution, TV window becomes
a red window after go to another page

===== Opened by (tester name) on (date) 05:20PM =====

Steps:

1. Make an image for 640x480 resolution
2. Load the build
3. Press F6
4. Then press F9 to return homw page

Result:

TV window become a red window.

Note:

1. It is only repro under 640x480. It is for NTSC TV.

Accroding to(a Dev name), it is a bug of Directx and related
with bug #7217.

===== Edited by (the tester name) on (date) 05:26PM =====

If it is bug of DirectX, I suggest to postpone it and find
a solution in driver as Feng did for 720x540.

===== Edited by feng on (same date) 06:56PM =====

I modified dshow, overlay mixer. When colorkey is pink,
I directly return pink value.



```

===== Resolved as Fixed by feng in 02.11.01.1356 on
         12/24/#### 02:12PM =====
--> AssignedTo: feng -> (tester name)
Temp solution. In halsurf.cpp, add this.
#ifdef IGS640x480
    ...
#endif

So we fix colorkey to 0xf81f (pink).
===== Closed by (tester name) on 12/24/#### 03:23PM =====
--> AssignedTo: (tester name) -> Closed
Verified it with VAK 1355.

```

附件 10.7 一个复杂的 Bug 报告

7187 Bug: BCS: EPG: Stress: Low memory appears and whole system is hangup when running stress epq testing page for 3 hours or so.

===== Opened by song on 12/04/#### 07:26PM =====

[Title]

When do stress testing for EPG about 3 hours, it will cause system low memory. The stress testing is:

1. Cycling set the time from monday to sunday.
2. Calling the method schedulentries.

[Steps To Repro]

1. Launch ...2.0 build 1338.
2. Navigate to website: http://...htm

[Observed Results]

OOM appears.
System is hangup.

[Note]

This bug maybe be caused by ...

===== Attached file "I:\CEPC\bin\oom.bmp" was added by
song on 12/04/#### 07:25PM =====

===== Attached file "C:\Temp\kato.log" was added by song
on 12/04/#### 07:25PM =====

===== Attached file "C:\Temp\stress.txt" was added by
song on 12/04/#### 07:25PM =====

===== Edited by yun on 12/04/#### 08:32PM =====

It is related with #6962 which was postponed.

===== Assigned by hui on 12/07/#### 02:56PM =====

--> AssignedTo: ... Bug Court -> fang

Pls. investigate this bug.

===== Edited by fang on 12/07/#### 10:28PM =====

I write a test application according the test page.

I find even the following call will cause memory leak.

...;

I will continue investigating it.

Fang



```

===== Edited by song on 12/10/#### 09:58AM =====
Below test case will cause system memory leak:
HRESULT ... (0)
{
    int nTotal=1000; //total repeated testing results

    ...;

    ...;

    ...;

    (Deleted related source codes)
    }

}

else

    (Deleted related source codes)
}

===== Resolved as Fixed by fang in 02.11.01.1346 on
12/11/#### 05:44PM =====
--> AssignedTo: fang -> song
This is a bug of EPD control
Fang

===== Activated by song on 12/12/#### 02:22PM =====
--> AssignedTo: song -> fang

When running the C-Code stress testing in build 1246 for
about 10 minutes, the low memory appears in e-talk.
===== Assigned by hui on 12/12/#### 05:58PM =====

```

--> AssignedTo: fang -> i-li
Pls. take a look at this bug.
==== Resolved as Fixed by i-li in 02.11.01.1347 on
12/12/#### 07:23PM ====
--> AssignedTo: i-li -> song
For one category, no memory leak. In normal situation,
the customer hard to cause low memory, only search
category by one day.
==== Edited by song on 12/13/#### 10:18AM ====
For six categories and search by one day, calling this
API for 12888 times will cause low memory, see attached
test page and test log. But I think it is hard to cause
low memory for one customer in reality. So I agree to
close this bug. Thanks.
==== Attached file "_\BCS\EPG\StressEPG.htm" was
added by song on 12/13/#### 10:17AM ====
==== Attached file "C:\Temp\stress.log" was added by
song on 12/13/#### 10:17AM ====
==== Closed by song on 12/13/#### 10:18AM ====
--> AssignedTo: song -> Closed
==== Activated by song on 12/13/#### 07:31PM ====
--> AssignedTo: Closed -> i-li
Guo, I have to re-active this bug again according to
leader's opinion. Could you please investigate it
again?
The testing result is:
When running below testing page about 4 hours or so,
the low memory appears and at the same time AVm shell
exception and IECE exception appears.

The testing page is : \\...\Stress\Result\StressEPG.htm

===== Resolved as Fixed by i-li in 02.11.01.1349 on
12/17/#### 02:19PM =====

--> AssignedTo: i-li -> song

Fix memeory leak in ... and ...

===== Activated by song on 12/18/#### 06:25PM =====

--> AssignedTo: song -> i-li

When running \\...\Stress\Result\StressEPG.htm and
sepg.exe (test_ForCategories) both cases will cause
avm exception.

===== Assigned by i-li on 12/18/#### 08:23PM =====

--> AssignedTo: i-li -> fang

Avmshell exception is cause by the release of Program.
Please investigate it.

===== Resolved as Fixed by fang in 02.11.01.1352 on
12/18/#### 08:43PM =====

--> AssignedTo: fang -> song

I can not find the exception when run ...

I can only repro this bug when run ...htm.

This because the call order of ... () and ... () is opposite.

The real bug is Program::... () do not clear the static
member ...

My last fix can escape the bug when run ...exe, but will
cause exception when run test page.

So my last fix is wrong to current design. I revert it.

I have modify the code of Program::... () to clear ...

Thanks

第 10 章



如何撰写测试文档
Write Good Testing Documents

Fang

===== Closed by song on 12/19/#### 10:00PM =====

--> AssignedTo: song -> Closed

See attached msg.

===== Attached file "C:\Temp\RE How about to repro the
bug #7187 using C coding.msg" was added by song on
12/19/#### 10:00PM =====

附录 A 微软亚洲研究院介绍

Appendix A The Introduction of Microsoft Research Asia

1998 年 11 月 5 日, 微软公司投巨资在北京设立微软中国研究院, 并于 2001 年 11 月 1 日宣布正式将其更名为微软亚洲研究院。

微软亚洲研究院是微软公司在海外开设的第二家基础科研机构, 也是该公司目前在亚太地区所设立的惟一的一所基础科研机构。这一战略投资显示了微软公司对中国及整个亚太区经济发展潜力的巨大信心以及对该地区信息产业发展的郑重承诺。微软公司希望通过此次更名, 将微软中国研究院三年来在吸引人才及科研方面所取得的一些好的经验更广泛地推广到亚洲其他国家和地区, 从而最大限度地提高该地区计算机基础科研的水平, 并满足微软在新世纪里不断创新和发展的需要。

自 1998 年 11 月 5 日成立以来, 微软亚洲研究院发展极其迅速。目前已经拥有在数字多媒体、多通道用户界面、无线及网络和亚洲信息处理技术方面的 120 多位优秀的科研技术人员。他们中的领军人物许多是从海外归来的、在各自的学术领域有很高造诣的年轻学者。研究院成立三年多来共在国际一流的学术刊物和会议上发表论文将近 400 篇, 申请国际专利逾百项, 并已有多项技术成功转移到微软公司的产品当中, 例如近期发布的 Office XP 和 Windows XP 等。

微软亚洲研究院还特别设立了“微软专家顾问委员会”, 聘请了浙江大学的潘云鹤校长、北京大学的迟惠生校长、清华大学的张钹教授、美国麻省理工学院的 Victor Zue 教授、香港科技大学名誉教授刘明雷先生和西安交通大学郑南宁副校长六位计算机研究领域的中

附录 A

附录 A 微软亚洲研究院介绍

Microsoft Research Asia
The Introduction of
Microsoft Research Asia





外资深学者担任顾问，对研究院的研究方向、整体运作进行指导，使研究院的研究在保持世界领先水平的同时，满足亚洲用户对技术的特别需求。

微软亚洲研究院现任院长兼首席科学家为张亚勤博士。张宏江博士和沈向洋博士任微软亚洲研究院副院长。

☛ 神圣的使命

微软研究院的使命是使未来的计算机能够看、听、学，能用自然语言与人类进行交流。在此基础上，微软亚洲研究院还将以更大的热情、付出更多的努力，促进计算机在亚洲地区的普及，改善亚洲用户的计算体验。

☛ 科研方向

微软亚洲研究院将主要从事以下四个方向的研究：

新一代用户界面：开创新的技术，让人们能够用更自然，更多元的方式和机器“交谈”，让使用计算机像与人交谈一样自然。研究院目前有两个小组正从不同的方向开展这方面的研究，其中，多通道用户界面组由王坚博士负责；语音技术组由张益肇博士负责。

新一代多媒体：开创新的技术，让多媒体能自动适应环境，拥有互动式操作，并在网上快速可靠地传送。互联网将成为新一代多媒体的中心。研究院目前有五个小组正从不同的方向开展这方面的研究，其中，网络多媒体组由李世鹏博士负责；多媒体计算组由张宏江博士负责；多媒体管理组由马维英博士负责；形象计算组由沈向洋博士负责；网络图形组由郭百宁博士负责。

新一代信息处理技术：开创新的技术，让亚洲人将来使用中文计算机像美国人使用英文计算机一样轻松、方便。研究院目前有自然语言组正在开展这方面的研究，该小组由周明博士负责。

新一代无线互联技术：开创新的技术，将 PC 时代强大的计算能力及多媒体应用扩展到无线网络环境中，实现真正的端对端的服务。研究院目前有无线网络组在开展这方面的研究，该小组由朱文武博士负责。



为更好地支持各研究小组的工作，研究院还成立了由软件开发工程师和技术支持工程师组成的新技术开发部，林斌任该开发部经理。

☞ 科研的环境

微软亚洲研究院提倡开放、自由、平等的学术风气，承诺为研究人员提供丰富的研究资源和长期的支持，鼓励研究人员要有长远的眼光和富于冒险的精神。对内，我们每一个人都深深地互信、互助、互重；对外，我们鼓励研究员们加强与外界的交流，发表我们的研究成果，并与国内外的高校及研究机构的学者们一起交流和探索研究项目。

☞ 架起学术交流的桥梁

微软亚洲研究院目前与包括中国大陆在内的亚洲各个国家和地区及世界各地的计算机科研机构进行着非常广泛的交流和合作，积极支持、推动和帮助最先进的计算机科学与技术在中国及整个亚洲地区的开发和应用。

微软亚洲研究院广泛参与国际一流的学术会议，积极投身相关的学报及期刊的编委会和学术指导委员会，并在重要的国际学术会议和刊物上发表研究成果。同时，微软亚洲研究院也在与全球一些优秀的大学合作，联合从事科研项目，支持博士后开展研究，并接待来自世界各地的访问学者。

微软亚洲研究院还特别在中国和部分亚洲国家和地区设立了“微软学者”奖学金机制，对当地优秀学生进行资助，使他们可以在良好的学术环境中完成学业并进行研究。

微软亚洲研究院将继续努力，为满足亚洲特别是中国市场在未来 5~10 年对于计算技术的需求奠定坚实的科研基础。微软亚洲研究院将在力所能及的范围内，积极配合亚洲各国政府的科技产业政策，最终为促进信息产业和互联网技术在亚洲地区的发展、推动整个地区的技术创新和进步做出贡献。



附录 B 课 程 设 计

Appendix B Project Design

题目设计者：微软亚洲研究院

附 录 B

附录 B 课程设计

课
程
设
计

B.1 实验

Project Name Speech Technology Application

Development platform Windows98/NT/2000

Development Tools SAPI 5.0 SDK, Platform SDK, and
Visual C++ 6.0

Requirements

- < Interest in applying speech recognition and speech synthesis technology in interesting applications
- < Design an application which uses speech technologies in interesting ways, e.g. voice interface to Age of Empire.
- < Machines with 128 MB of RAM and PII 233 MHz or above.
See <http://www.microsoft.com/speech/SpeechSDK/readme.htm> for details.
- < Build up a demo including friendly UI, robust system, nice output





format of result, etc.

- ◀ Submit a design report, including motivation for the application, application design, source code, code description document, test report, and demo. Usability studies will be a plus.
- ◀ Since SAPI 5.0 SDK can be freely redistributed, it will be possible to share the final application with your friends!

Resource

www.microsoft.com/speech

SAPI 5.0 Help Files

SAPI 5.0 Sample Applications

B.2 题目

Project Name Video Segmentation

Development platform Windows98/NT/2000

Development Tools DirectX Media SDK6.0, Visual C++
6.0

Requirements

- ◀ Design an algorithm for shot change detection (or other Video/Audio processing). The shot change includes abrupt



transitions (cut), gradual transitions (dissolve, fade in, fade out), and wipes.

- ◀ Design a DirectX Media filter to implement the algorithm;
- ◀ Build up a demo for filter.
- ◀ Evaluate the performance of the algorithm.
- ◀ Do some analysis on your experiment results;
- ◀ Submit a technical report, including algorithm description, source code, code description document, result analysis, and demo.

Resource

◀ Reference

1. DirectX Media SDK (<http://www.microsoft.com/directx/dxm/>)
2. Hong-Jiang Zhang, Atreyi Kankanhalli, Stephen W. Smoliar, "Automatic partitioning of full-motion video", Multimedia system, p.10-28, 1993.

◀ Test Data

\\MSRCNMC\MC\mpeg7\cd21\misc2.mpg or any other legal MPEG-3 video streams.





Resource

< Reference

1. DirectX Media SDK (<http://www.microsoft.com/directx/dxm/>)
2. Hong-Jiang Zhang, Atreyi Kankanhalli, Stephen W. Smoliar, “Automatic partitioning of full-motion video”, Multimedia system, p.10-28, 1993.

< Test Data

\\MSRCNMC\MC\mpeg7\cd21\misc2.mpg or any other legal MPEG-3 video streams.



编辑手记

2001 年夏秋之交，我和我的同事施玉新参加微软亚洲研究院为高校老师所做的操作系统培训。我们有机会参观了微软（中国）公司和微软亚洲研究院。微软亚洲研究院商务及高校关系部高级经理陈宏刚博士给到会的教师做了关于微软公司和微软亚洲研究院的情况的报告。在陈博士精彩的报告中，提到一件事：微软亚洲研究院为提高中国学生的开发素质，在北京大学以及其他一些著名高校开设了《软件开发的科学与艺术》的系列讲座。在高校中引起了强烈的反响。

作为 IT 出版人，我的头脑中迸发出这样一个想法——如果能够将这些精彩讲座整理成书，将会使所有探索软件开发的学生、教师及其他从业人员受益！

因为播放 PowerPoint，所有的灯光都关闭了，我摸着黑把与这个线索有关的情况记录在纸上。培训结束的第二天，我们与亚洲研究院的陈宏刚博士取得了联系，并初步交换了意见。随后，在短短 2 个小时的面谈中，我们已经把整理本书的相关事宜基本确定。在接下来的几个月时间里，在我们和研究院的陈宏刚、马歆等人的沟通、组织下，书稿一点点见了眉目。

本书是在各位微软专家的讲演稿基础上形成的，包含了微软专家多年来凝聚的开发经验，句句是肺腑之言，而且各位专家在百忙的工作中对本书稿的每一个细节进行了认真的审定。所以，首先要感谢这些与中国软件从业和学习人员进行无私交流的微软专家们。

在本书的整理过程中，得到了微软亚洲研究院（MSRA）的全力支持和配合。张亚勤院长、陈宏刚博士、林斌经理亲自参与本书的写作，并且及时解决了本书出版过程中所面临的困难。高校关系部技术专员马歆小姐在长达 8 个月的时间里，一直事无巨细而井然有序地为本书的出版忙碌着。如果把本书的编写成员和组织成员看成



一个虚拟团队（Virtual Team）的话，可以这样说，她灿烂的笑容和爽朗的声音为这个 Team 带来了青春的活力。

MSRA 高校关系部的很多同事都参与了本书出版的相关工作，他们是张高博士、刘佐扬小姐、张煜先生等人。微软公司专门负责出版业务的孙文清先生也曾经为本书的出版提出非常中肯的意见。一次愉快的合作已经使我们成为很好的朋友。

与微软亚洲研究院高校关系部的各位朋友的合作，让我们深刻体会到微软公司的雷厉风行、严谨求实、富有创新精神的风格和企业文化。整个合作过程为我们准确把握本书的立意提供了很好的注脚。



微软亚洲研究院高校关系部的成员

（左起：张煜、刘佐扬、陈宏刚、马歆、张高、郑薇、王瑾）

在本书的整理过程中，俞俊平、余安萍等人做了艰苦而细致的工作。为按时完成本书的整理工作，他们度过了很多个不眠之夜。没有他们的努力，本书的出版只能是空谈。

本书的成功面世，要感谢许多人的支持与协作。2002 年 1 月，

在《程序员》杂志和 www.csdn.net 的支持下，我们开展了征集书名的活动。本书出版前，一部分读者也在 www.csdn.net 等网站上预览了本书的部分章节。另外，近 1 年来，很多从事软件开发工作的朋友对本书的关注和支持也是我们作好本书的动力，他们是北京的周长发、陆杰，广东的孙伟，湖南的伯晓晨、应晓敏，江苏的段钢等人。

本书的出版是与电子工业出版社各级领导和同事的支持分不开的。特别是在出版之前的一个月，美编室、排版室、印务部的同事与我们在共同煎熬中度过，发行部的同仁则为本书的宣传和推广做了许多扎实的工作。

作为本书的组织者，我和我的同事施工新享受了它的整个运作过程——虽然非常辛苦，但绝对是享受。工作着是快乐的，更何况是在为这样一本美妙的图书的面世做准备。我们当然也是本书的第一读者，先于大家体味了本书的精美内容。

本书虽然由多位作者分别完成，但是写作浑然一体，可以一气呵成地阅读，也可以作为教材由老师讲解。当然，它更适合在工作和学习的时间信手拈来，仔细品味。好茶过了两道才更有味道，一本好书亦是如此。

认真体会本书内容，你的软件开发功底在潜移默化中就已经提高。身处在这这么一个竞争激烈而日新月异行业中，有机会站在巨人的肩膀上行事，何乐而不为？

十多年以来，我已经读了不知多少本计算机图书，做编辑工作的 7 年来，也编辑了百余本图书，然而，能留存至今仍然难于割舍的书已经不多了，想必从事软件开发工作的你与我有类似的做法。

希望本书是个例外。好书是经受岁月冲刷而沉淀下来的沙金，很重，不耀眼，却值得与你相伴永远。

你的感觉，请告诉我：guoli@phei.com.cn。

郭立

2002 年 4 月 北京



编辑手记
Editor's Voice



《软件开发的科学与艺术》读者调查表

尊敬的读者：

感谢您对我们的支持与爱护。您的意见是我们创造精品的动力源泉!为了今后为您提供更优秀的图书,请您抽出宝贵的时间将您的意见以下表的方式(亦可从 <http://jsj.phei.com.cn> 下载本调查表)及时告知我们。我们将从中评选出热心读者若干名,免费赠阅我们以后出版的图书。

姓名: _____ 性别: ☐ 男 ☐ 女 年龄: _____ 职业: _____
电话(寻呼): _____ E-mail: _____
通信地址: _____ 邮编: _____

1. 影响您购买本书的因素(可多选):

☐封面封底 ☐价格 ☐内容提要、前言和目录 ☐书评广告 ☐出版社名声
☐作者名声 ☐正文内容 ☐讲座及其他 _____

2. 您对本书的满意度:

从技术角度 ☐很满意 ☐比较满意 ☐一般 ☐较不满意 ☐不满意
☐改进意见 _____

从文字角度 ☐很满意 ☐比较满意 ☐一般 ☐较不满意 ☐不满意
☐改进意见 _____

从版面、封面设计角度 ☐很满意 ☐比较满意 ☐一般 ☐较不满意
☐不满意 ☐改进意见 _____

3. 您最喜欢书中的哪篇(或章、节)?请说明理由。

4. 您最不喜欢书中的哪篇(或章、节)?请说明理由。

5. 您希望本书在哪些方面进行改进?

6. 您感兴趣或希望增加的图书选题有:

请寄: 北京海淀区万寿路 173 信箱计算机事业部(100036)

电话: 010-68216158

E-mail: jsj@phei.com.cn

征集书评启事

在微软亚洲研究院和电子工业出版社计算机图书事业部的共同努力下,《软件开发的科学与艺术》一书终于面市了。事实上,在此之前,本书的 Powerpoint 演示文稿已在网上广泛公布,并得到了许多网友的反馈。

我们希望通过本书的出版以及后续的系列工作,建立起可供国内软件开发人员交流经验的平台。为此,特筹备《软件开发的科学与艺术》书评征集活动。

欢迎读者将您的读后感受与大家分享。书评活动说明如下:

1. 书评要求体现每个读者的切身感受,文字贵在简明而透彻,不要粉饰图书本身;应以交流读书体会,引导他人领会图书精髓为目的;字数、文体不限。

2. 可以将书评以下列方式提交。

(1) 100036 北京万寿路 173 信箱 计算机图书事业部 郭立(请注明“书评”二字);

(2) 发邮件至 guoli@phei.com.cn 或 i-xma@microsoft.com (信件主题请注明“书评”)。

3. 除特殊声明外,视作者同意将书评在相关网站和专业媒体上发表(署名方式由作者提供)。

4. 若书评被专业媒体采用,将支付相应稿酬。

5. 本活动共评选 20 篇优秀书评,结果将于 2002 年 7 月和 2002 年 12 月分两次揭晓,获奖书评作者将有机会参观微软(中国)有限公司和微软亚洲研究院,并获得颇具纪念意义的礼品和书籍。

电子工业出版社计算机图书事业部主办本活动,并拥有最终解释权。

电子工业出版社计算机图书事业部

2002.4